DEFENSE INFORMATION INFRASTRUCTURE (DII)

COMMON OPERATING ENVIRONMENT (COE)

PROGRAMMER'S REFERENCE MANUAL SET

VOLUME IV

JMTK Version 3.0 Developers Manual - Part 2.
(Man Pages)

28 June 1996

Prepared for:

Defense Information Systems Agency

TABLE OF CONTENTS

INTRODUCTION

This manual presents the JMTK APIs by functions performed. All information presented in this manual supports the ANSI C language. JMV_MDrawArc is an example of an API used in the JMTK Program. This naming format has been adopted by the JMTK Technical Working Group. The first two characters of each API are JM for Joint Mapping. The third character identifies the domain within JMTK according to the following scheme:

V	=	Visual
A	=	Analysis
S	=	Spatial Database Management
U	=	Utilities
I	=	Local Imaging Manipulations (future use)
G	=	Geographical Data Services (future use)
R	=	Security, Access, and Data Releasibility (future use)

The fourth character is an underscore _ . The fifth and all successive characters represent the API name. All legacy API names used in this manual are arranged in a verb followed by noun sequence. Example: MDrawArc. Future naming of APIs will transition to a noun-verb naming scheme. This verb-noun scheme will be incorporated in a subsequent revision of this material.

A complete list of JMTK APIs presented in this manual is found in Appendix A. Also included in the listing is the cross reference to the Software Requirements Specification (SRS) paragraph which established the functional API requirement.

Each section contains a brief overview of the API functional grouping. A sample of code using many of the APIs is in a separate volume. Each individual API is addressed with pertinent information formatted in categories which display individual API information as follows:

1.0 Function - contains a non technical description of what the API does.

2.0 Description - contains the technical description describing the program process and any associated steps in the process required to reach the functional goal.

3.0 Arguments - presents and details the actual logic process and parameters.

4.0 Returns - lists values returned upon completing the function.

5.0 Dependencies - lists dependent APIs.

6.0 Messages - lists displayed messages upon failure to execute.

7.0 Other APIs - lists other related APIs.

8.0 Related Documentation - describes any documents which may be related to the API.

SECTION 1
Symbol Manipulation

The APIs in this section are legacy code and are used for manipulating symbology and setting attributes. The following list of APIs represents the Section 1, Symbol Manipulation, capabilities for the JMTK:

API Name	*Req. Para.*
JMV_MApplyAttributes	2.10
JMV_MApplyData	2.10
JMV_MApplyFillOffset	2.4.10
JMV_MApplyFillType	2.4.10
JMV_MApplyFillWeight	2.4.10
JMV_MApplyVisibility	2.10
JMV_MCopyTemplate	2.10
JMV_MModifyObject	2.10
JMV_MQueryObject	2.10
JMV_MQueryObjectBBox	2.10
JMV_MSetAttributes	2.10
JMV_MSetData	2.10
JMV_MSetFillOffset	2.4.10
JMV_MSetFillType	2.4.10
JMV_MSetFillWeight	2.4.10
JMV_MSetObjectData	2.10
JMV_MSetVisibility	2.10

SECTION 2
Drawable Display Objects

The APIs in this section are legacy code and are drawable display objects created using the MDraw commands. These routines provide interface with the JMTK for creating graphical objects. MDraw is the root command (verb) followed by an object name (noun) describing the object to be drawn. The following display objects are included in this section:

Arc	Poly
Box	Polygon
Char	Rectangle
Char16	Sector
Circle	Segment
Ellipse	Slash
Line	Symbol
Polyline	Text

These objects must have input from the user in order to determine the shape, size, or placement of the object in the window.

In some cases a template must first be created in order to establish a place to install the required input parameters.

The user is responsible for creation of a template of object attributes. This template is then applied to the window in use, thus enabling creation of the initial object attributes. Since this window is associated with the supplied template parameter, the window id is not part of the MDraw object routine.

The following list of APIs represents the Section 2, Draw Objects, capabilities for the JMTK:

API Name	*Req. Para.*
JMV_FormLine	2.10.1.12
JMV_FormPolygon	2.10.1.8
JMV_FormPolyLine	2.10.1.11
JMV_InitializePointSet	2.10.1.2
JMV_MChangeText	2.10.1.11
JMV_MCreatePoly	2.10.1.8
JMV_MCreateText	2.10.1.11
JMV_MDrawArc	2.10.1.4
JMV_MDrawBitmap	2.10.4.3
JMV_MDrawBox	2.10.1.6
JMV_MDrawChar	2.10.1.2
JMV_MDrawChar16	2.10.1.2
JMV_MDrawCircle	2.10.1.3

JMV_MDrawEllipse	2.10.1.5
JMV_MDrawLine	2.10.1.12
JMV_MDrawPolygon	2.10.1.8
JMV_MDrawPolyLine	2.10.1.1
JMV_MDrawRectangle	2.10.1.6
JMV_MDrawSector	2.10.1.2
JMV_MDrawSegment	2.10.1.10
JMV_MDrawSlash	2.10.1.12
JMV_MDrawSymbol	2.10.1.4
JMV_MDrawText	2.10.1.11
JMV_MSetSegment	2.10.1.1
JMV_SetPointInterpolation	1.5

SECTION 3
Display Settings

The APIs in this section are legacy code. They are capable of displaying, in text fields and formats, data associated with a symbol's specified pixel offset or another georeferenced symbol within a map layer. Two of the APIs specifically support and manage symbology libraries. The graphic objects included are:

Text Size	Color
Font	Outlining
Style	Thickness
Orientation	Blinking
Brightness	

The following list of APIs represents the Section 3, Display Settings, capabilities for the JMTK:

API Name	*Req. Para.*
JMU_MuPointSizeToFontName	2.4
JMU_MuPointSizeToFontWidth	2.4
JMV_MApplyFont	2.10.10
JMV_MApplyHiLite	2.10.11
JMV_MApplyLineStyle	2.10.10
JMV_MApplyLineType	2.10.10
JMV_MApplyLineWidth	2.10.10
JMV_MApplyTemplate	2.10.10
JMV_MSetFont	2.10.10
JMV_MSetHiLite	2.10.11
JMV_MSetHiLiteColor	2.10.11.2
JMV_MSetLineStyle	2.10.10
JMV_MSetLineType	2.10.10
JMV_MSetLineWidth	2.10.10
JMV_MSetTemplate	2.10.10

SECTION 4
Display Features

The APIs in this section are legacy code and are capable of hiding or unhiding features from a particular data source, symbols or map layers, and other pertinent data. The following list of APIs represents the Section 4, Display Features, capabilities for the JMTK:

API Name	*Req. Para.*
MU_MuFeatureToAcronym	util
JMU_MuFeatureToString	util
JMV_AppendLine	2.10.6
JMV_AppendPolygon	2.10.6
JMV_AppendPolyLine	2.10.6
JMV_MAddFeature	2.10.8.3
JMV_MAddFeatures	2.10.8.4
JMV_MAddPoint	2.10.8.3
JMV_MApplyColor	2.10.7.9
JMV_MApplyPickability	2.10.8
JMV_MApplyPixel	2.10.7.9
JMV_MCreateObject	2.10.6
JMV_MCreateTemplate	2.10.8.3
JMV_MDrawWeather	2.10.6
JMV_MSetColor	2.10.7.9
JMV_MSetIntensity	2.10.7.9
JMV_MSetIntensityDetail	2.10.7.9
JMV_MSetIntensityModels	2.10.7.9
JMV_MSetMapColors	2.10.7.9
JMV_MSetMapColorsByRGB	2.10.7.9
JMV_MSetPickability	2.10.8
JMV_MSetPixel	2.10.7.9
JMV_MUpdateClass	2.10.8

SECTION 5
Edit Features

The APIs in this section are legacy code and are capable of moving an object from a window, modifying one or a list of feature attributes on a map, and removing objects from a list or map products and features from the Draw Module or geographic display. The following list of APIs represents the Section 5, Edit Features, capabilities for the JMTK:

API Name	*Req. Para.*
JMV_ClearPointSet	2.11.1
JMV_FreePointSet	2.11.1
JMV_GenRemoveFeatures	2.11.1.1
JMV_GenRemoveProducts	2.11.1.1
JMV_MModifyFeature	2.11.1.7
JMV_MModifyFeatures	2.11.1.7
JMV_MMoveObject	2.10.7.1
JMV_MRemoveFeature	2.11.1
JMV_MRemoveFeatures	2.11.1
JMV_MRemoveObject	2.11.1.1
JMV_MRemoveProduct	2.11.1.1
JMV_MRemoveProducts	2.11.1.1

SECTION 6
Data Transformations

The APIs in this section are legacy code and are capable of converting a standard set of unit conversions. Some retrieve the projection data structure in use for the window coordinate system and others actually convert geodetic coordinates (lat/long) to point-to-pixel coordinates and vice versa. The following list of APIs represents the Section 6, Data Transformations, capabilities for the JMTK:

API Name	*Req. Para.*
JMV_ComputeScales	3.6
JMV_DegreesToRadians	2.2.2
JMV_FeetToMeters	2.2
JMV_KilometersToMiles	2.2
JMV_KilometersToNautical	2.2
JMV_MetersToFeet	2.2
JMV_MetersToNautical	2.2
JMV_MGetProjectionData	3.6
JMV_MilesToKilometers	2.2
JMV_MilesToMeters	2.2
JMV_MilesToNautical	2.2
JMV_MPixelsToPosition	2.2.12
JMV_MPositionToPixels	2.2.18
JMV_NauticalToDegrees	2.2
JMV_NauticalToKilometers	2.2
JMV_NauticalToMeters	2.2
JMV_NauticalToMiles	2.2
JMV_PixelsToPosition	2.2.126
JMV_PositionToPixels	2.2.18
JMV_RadiansToDegrees	2.2.11

SECTION 7
Windows

The APIs in this section are legacy code and are capable of providing a consistent interface to the Chart Manager and to various Windows from disparate Chart Clients. The different APIs perform tasks such as mapping a window to the screen and destroying a window in the chart manager. The following list of APIs represents the Section 7, Windows, capabilities for the JMTK:

API Name	*Req. Para.*
JMU_MuInitialize	util
JMU_MuTerminate	util
JMV_GenReserveFeature	3.1
JMV_GenReserveMap	util
JMV_MAbortMap	2.3
JMV_MAddObject	2.3.5
JMV_MAddProduct	2.3.5
JMV_MAddProducts	2.3.5
JMV_MChangeMap	2.3
JMV_MCreateClass	2.3.5
JMV_MCreateList	2.3.5
JMV_MCreateWindow	2.3.1
JMV_MDestroyList	2.3.1.3
JMV_MDestroyObject	2.3.1.3
JMV_MDestroyWindow	2.3.1.3
JMV_MDrawWorld	2.3.1.6
JMV_MExchangeObject	2.3.5
JMV_MGetXWindow	2.3
JMV_MMapWindow	2.3.1.6
JMV_MReleaseWindow	2.3.1.3
JMV_MReorderMaps	2.3.5.3
JMV_MResetIntensity	2.3.8.2
JMV_MUnMapWindow	2.3.1.4
JMV_MUseNamedWindow	2.3.1
JMV_MUseWindow	2.3.1

SECTION 8
Display View

The APIs in this section are legacy code and are capable of displaying various views of the current map and its display boundaries and width along with changing the mode of the cursor. Some of these functions include: quickly zooming the current map view, and recentering and/or rescaling a map. The following list of APIs represent the Section 8, Display View, capabilities for the JMTK:

API Name	*Req. Para.*
JMV_MQuickZoom	2.8.1
JMV_MRecenterMap	2.8.3
JMV_MScaleMap	2.8.1
JMV_MSetCursorMode	2.8
JMV_MSetMapBounds	2.8.2
JMV_MSetMapWidth	2.8.2

SECTION 9
SDB Retrieval

The APIs in this section represent new and legacy code. The new APIs are marked. They are capable of various procedures regarding the Spatial Data Base. Some APIs specifically address accessing the SDB data and terminating previous connections. Others establish unique connections and define geographic AOIs (areas of interest). Many of these APIs perform retrieving functions and one saves matrix data in a SDB. The following list of APIs represents the Section 9, Spatial Data Base Retrieval, capabilities for the JMTK:

API Name	*Req. Para.*	*New APIs*
JMS_ConfigAOIGet		new
JMS_ConfigAOISet		new
JMS_DataPathnameGet		new
JMS_DbConnect		new
JMS_DbDisconnect		new
JMS_DbListGet		new
JMS_ErrorGet		new
JMS_InventoryGet	3.4.9	new
JMS_MatrixGet		new
JMS_MatrixPut		new
JMS_MetadataGet	3.3	new
JMV_MAddVolume	3.4.9	
JMV_MListMaps	3.4.9	
JMV_LoadMap		new
JMV_MReloadSearchPath	3.4.9	
JMV_MRemoveVolume	3.4.9	
JMV_UnLoadMap		new

SECTION 10
Display Utilities

The APIs in this section are legacy code and are capable of the following functions:

- Abort the animation of an object
- Add an input source to the context
- Register a time-out with the service manager
- Flush the output buffer
- Flush the event queues for all channels
- Get the next event from the event queue
- Return the number of pending Map events
- Push an event back on the input queue
- Release point select focus
- Remove an input source from the Chart Manager
- Remove an interval timer
- Request map focus
- Change cursor to the normal cursor
- Send a map event to other users of a window
- Set the control keys used during animation

The following list of APIs represents the Section 10, Display Utilities, capabilities for the JMTK:

API Name	*Req. Para.*	
JMV_MAbortAnimation	util	
JMV_MAddInput	util	
JMV_MAddTimeOut	util	
JMV_MFlush	util	
JMV_MFlushAllEvents	util	
JMV_MNextEvent	util	
JMV_MPending	util	
JMV_MPutBackEvent	util	
JMV_MReleaseFocus		util
JMV_MRemoveInput		util
JMV_MRemoveTimeOut	util	
JMV_MRequestFocus	util	
JMV_MRestoreCursor	util	
JMV_MSendEvent	util	
JMV_MSetAnimateKeys	util	

SECTION 11
Coordinate Transformations

The APIs in this section are legacy code and are discussed in the following general man pages.

The *MuConvert* utilities provide a set of routines for converting among three map coordinate systems: Geodetic (GEO), Universal Transverse Mercator (UTM), and Military Grid Reference (MGR). The *MuGeoPosn* routines and the *MuPosition* routines which provide conversions from geodetic values to a printable string, and vice-versa. The *MuMgrPosn* routines provide conversions from *MGR COORD* records to a printable string, and vice-versa. The *MuUtmPosn* routines provide conversions from UTM values to a printable string, and vice-versa. These API man pages have been included in this section.

The following list of APIs represents the Section 11, Coordinate Transformations, capabilities for the JMTK:

API Name	*Req. Para.*
JMU_MuGeoMgr	2.2
JMU_MuGeoPosition	1.3.5.2
JMU_MuGeoToString	2.2.4
JMU_MuGeoUtm	2.2.6
JMU_MuMgrGeo	2.2
JMU_MuMgrToString	2.2
JMU_MuMgrUtm	2.2
JMU_MuPositionToString	2.2
JMU_MuStringToPosition	util
JMU_MuUtmGeo	2.2.10.
JMU_MuUtmMgr	2.2
JMU_MuUtmToString	2.2
JMU_MuValidGeo	2.2
JMU_MuValidMgr	2.2
JMU_MuValidUtm	2.2

SECTION 12
Symbol Library - Add to List

The APIs in this section are legacy code and are discussed in the following general man pages.

MuReference includes Chart Manager category reference routines. The *MapFeatureAttributes (MFeatAtts)* structure is used to describe the list of features which are drawn onto a map window. These API man pages are included in this section.

The following list of APIs represents the Section 12, Symbol Library - Add to List, capabilities for the JMTK:

API Name	*Req. Para.*
JMU_MuAddFeature	2.3.6.4
JMU_MuAddProjection	3.4
JMU_MuAddSubFeature	2.10
JMU_MuAddSubType	2.10.8.3
JMU_MuAddType	3.4
JMU_MuListFeatures	3.10.15
JMU_MuListProjections	3.10.11
JMU_MuListSubFeatures	3.10.40
JMU_MuListSubTypes	3.10.35
JMU_MuListTypes	3.10.35
JMU_MListFeatures	

SECTION 13
Analysis

The APIs in this section are revised legacy code and are not available as of 10 April 1996.

The following list of APIs represents the Section 13, Analysis, capabilities for the JMTK:

API Name	*Req. Para.*
JMA_FanAnalyze	1.2
JMA_ProfileAnalyze	1.3

SECTION 14
Error

The APIs in this section are legacy code and are discussed in the following general man pages.

MuError API includes the Chart Manager standardized error and warning utilities. *GenError* API includes the Draw Module error handling routines. The *GenSendError* routine sends an error to the Chart Manager.

The following list of APIs represents the Section 14, Error, capabilities for the JMTK:

API Name	*Req. Para.*
JMU_MuAppError	util
JMU_MuAppWarning	util
JMU_MuErrorHandler	util
JMU_MuErrorMsg	util
JMU_MuSetErrClass	util
JMU_MuSetErrorList	util
JMU_MuSysError	util
JMU_MuSysWarning	util
JMV_GenErrorToString	util
JMV_GenRequestCodeToString	3.1
JMV_GenResetErrorHandler	util
JMV_GenResetIOErrorHandler	util
JMV_GenSendError	util
JMV_GenSetErrorHandler	util
JMV_GenSetIOErrorHandler	util
JMV_MErrorToString	util
JMV_MMajorCodeToString	util
JMV_MMinorCodeToString	util
JMV_MResetErrorHandler	util
JMV_MResetIOErrorHandler	util
JMV_MSetErrorHandler	util
JMV_MSetIOErrorHandler	util

SECTION 15
Display Query

The APIs in this section are legacy code and are capable of the following functions:

- Chart Manager distance utility routines.
- List the features available on the Chart Manager.
- Retrieve display list of features for given geographic display.
- Get current geographic display attributes for a given map window.
- Get information about a Map Window.
- Set the annotation for the cursor, when in normal cursor mode.

The following list of APIs represents the Section 15, Display Query, capabilities for the JMTK:

API Name	*Req. Para.*
JMU_MuBearing	3.10.81
JMU_MuDistance	3.10.81
JMU_MuGetPosition	1.3.5.2
JMU_MuGetRangeBearing	2.15.1.6
JMU_MuTargetAltitude	3.10.65
JMU_MuTargetMaxRange	3.10.65
JMV_MuListFeatures	3.10.15
JMV_MQueryFeatures	3.10.15
JMV_MQueryMap	3.10.9
JMV_MQueryWindow	3.10.11
JMV_MSetCursorAnnotation	2.15

SECTION 16
Symbol

The APIs in this section are legacy code and consist mainly of Chart Manager category reference routines and option parsing routines. A smaller portion of the APIs in this section include the following Chart Manager routines: geo reference position string utility routines, MGR position string utility routines, and Chart Manager UTM position string utility routines.

The following list of APIs represents the Section 16, Symbol, capabilities for the JMTK:

API Name	*Req. Para.*
JMU_MuOption	
JMU_MuUtmPosn	
JMU_MuProjectionToAcronym	util
JMU_MuProjectionToString	util
JMU_MuQualGetOption	util
JMU_MuQualGetPgmName	util
JMU_MuQualUsage	util
JMU_MuStringToFeature	util
JMU_MuStringToGeo	util
JMU_MuStringToMgr	util
JMU_MuStringToProjection	util
JMU_MuStringToSubFeature	util
JMU_MuStringToSubType	util
JMU_MuStringToType	util
JMU_MuStringToUtm	util
JMU_MuSubFeatureToAcronym	util
JMU_MuSubFeatureToString	util
JMU_MuSubTypeToAcronym	util
JMU_MuSubTypeToString	util
JMU_MuTypeToAcronym	util
JMU_MuTypeToString	util

SECTION 17
Windows - Communication

The APIs in this section are legacy code. These APIs perform functions such as opening and closing a communications channel to the Chart Manager, notifying Chart Manager of various Draw Module activities, flushing the Draw Module request queue, and shutting down the Chart Manager.

The following list of APIs represents the Section 17, Windows - Communication, capabilities for the JMTK:

API Name	*Req. Para.*
JMV_GenAddFeatures	2.10.18.3
JMV_GenAddProducts	2.3.5
JMV_GenAttach	2.3
JMV_GenChangedFeature	2.3.6.4
JMV_GenChangedMap	2.3
JMV_GenDetach	2.3
JMV_GenDrawingFeature	util
JMV_GenDrawingMap	util
JMV_GenFlushAllRequests	2.3
JMV_GenGetDisplay	2.3
JMV_GenInitialize	2.3
JMV_GenNextRequest	util
JMV_GenPending	util
JMV_GenServerToSocket	util
JMV_MChannelToSocket	util
JMV_MCloseChannel	util
JMV_MKillServer	util
JMV_MMainLoop	util
JMV_MNoOp	util
JMV_MOpenChannel	util
JMV_MQueryChannel	util
JMV_MSetEventHandler	util
JMV_MSetEventMask	2.24
JMV_MSync	util

SECTION 18
Memory Manager

The APIs in this section are legacy code and consist mainly of Chart Manager memory allocation utilities and object search utilities.

The following list of APIs represents the Section 18, Memory Manager, capabilities for the JMTK:

API Name	*Req. Para.*
JMV_MAlloc+	util
JMV_MAllocVerify	util
JMV_MDebug	util
JMV_MFree	util
JMV_MGetSearchPath	util
JMV_MReAlloc	util

Alphabetical List of Man Pages

GenAddFeatures

FUNCTION

Specifiy map features which are of interest to this Draw Module.

SYNTAX

C Interface

```
void GenAddFeatures( server, products, numproducts ) ServerId       server;
                                                     FeatureProduct *products;
                                                     int            numproducts;
```

ARGUMENTS

server The link between the Draw Module and the Chart Manager to which it is connected. Returned by GenAttach().

products Describes a list of feature products which are of interest to this Draw Module.

numproducts

The number of products described in products.

DESCRIPTION

The GenAddFeatures() call specifies feature products that are of interest to this Draw Module. Specifying products of interest guarantees that the Draw Module will receive a FeatureVerifyRequest for each map file matching any of the requested products. The Draw Module is not obligated to support all products which it specifies in the product list. Rather, this list simply limits the number of FeatureVer- ifyRequests sent to it by the Chart Manager.

The value AnyFeature serves as a wildcard and is supported with any of the fields in the FeatureProduct structure, including the FeatureType field. The FeatureType, and FeatureSubType fields are the same fields used in the Map- FeatureAttributes structure.

After sending this response to the Chart Manager, the Draw Module should expect a FeatureVerifyRequest for each feature product in the system which are in the list. The Draw Module can make this call at any time, adding module support for other products at a subsequent time. The GenRemo- veFeatures() call is used for removing feature products of interest.

ERRORS

BadServer

An invalid server id was used.

BadValueError

An invalid product specification was sent to Chart Manager. The specified value for FeatureType or FeatureSubType is/are not supported by the Chart Manager.

SEE ALSO

GenAttach(3Gen), GenAddProducts(3Gen), GenRemoveFeatures(3Gen), GenFeatVerify(3Gen), MuReference(3Mu)

GenAddProducts

FUNCTION

Specify map products which are of interest to this Draw Module.

SYNTAX

C Interface

```
void GenAddProducts( server, products, numproducts ) ServerId    server;
                                                     MapProduct  *products;
                                                     int         numproducts;
```

ARGUMENTS

server The link between the Draw Module and the Chart Manager to which it is connected. Returned by GenAttach().

products Describes a list of map products which are of interest to this Draw Module.

numproducts

The number of products described in products.

DESCRIPTION

The GenAddProducts() call specifies map products that are of interest to this Draw Module. Specifying products of interest guarantees that the Draw Module will receive a Map- VerifyRequest for each map file matching any of the requested products. The Draw Module is not obligated to support all products which it specifies in the product list. Rather, this list simply limits the number of MapVerifyRe- quests sent to it by the Chart Manager.

The value AnyMap serves as a wildcard and is supported with any of the fields in the MapProduct structure, including the MapType field. The MapType, and MapSubType fields are the same fields used in the MapChangeAttributes structure.

After sending this response to the Chart Manager, the Draw Module should expect MapVerifyRequests for all map products in the system which are in the list. The Draw Module can make this call at any time, adding module support for other products at a subsequent time. The GenRemoveProducts() call is used for removing products of interest.

ERRORS

BadServer

An invalid server id was used.

BadValueError

An invalid product specification was sent to Chart Manager. The specified value for MapType or MapSubType is/are not supported by the Chart Manager.

SEE ALSO

GenAttach(3Gen), GenAddFeatures(3Gen), GenRemoveProducts(3Gen), GenMapVerify(3Gen), MuReference(3Mu)

GenAttach

FUNCTION

Attachs a Draw Module process to a Chart Manager.

SYNTAX

C Interface

```
ServerId GenAttach( host )
        char   *host;
```

ARGUMENTS

host	The name of the host where Chart Manager is executing.

DESCRIPTION

The GenAttach() routine attaches the Draw Module process to the Chart Manager on the specified host. A ServerId is returned and is used to reference the connection with the Chart Manager. Almost all calls made to the library require a ServerId as a parameter. If the library is unable to connect to the Chart Manager, then InvalidServerId is returned.

If the host field is a NULL pointer, then the environment variable MapHostName is checked for, and if it is defined then its value is used as the name of the host to connect to. If MapHostName is not a defined environment variable, then the Gen library attempts to connect to the local host.

RETURN

Upon successful connection, the ServerId of the connection is returned; otherwise InvalidServerId (-1) is returned.

ERRORS

AlreadyConnected

The Draw Module already has an open connection to this Chart Manager via a previous GenAttach() call. No more than one connection per Chart Manager is allowed for each Draw Module.

OutOfMemory

Unable to allocate space for this Chart Manager connection. No memory left.

SEE ALSO

GenDetach(3Gen), GenEnviron(3Gen), setenv(1)

GenChangedFeature

FUNCTION

Notify Chart Manager that the draw module has finished rendering a feature.

SYNTAX

C Interface

```
void GenChangedFeature( server)
        ServerId    server;
```

ARGUMENTS

server The link between the Draw Module and the Chart Manager to which it is connected. Returned by GenAttach().

DESCRIPTION

The GenChangedFeature() call notifies the Chart Manager that the feature which the draw module has been currently rendering has completed. This call should be sent by a Draw Module ONLY IF the module has just received a FeatureDrawRequest. Because of this, the implied parameters in the call reference the draw request currently being worked on.

When a Draw Module receives a FeatureDrawRequest, the module can either choose to ignore it, or render the feature as requested. If the latter action is done, the module needs to notify the Chart Manager as to its progress (via periodic calls to GenDrawingFeature()) and must indicate when the rendering has completed via a call to GenChangedFeature(). Calls to this routine at times other than when the Draw Module is currently rendering a feature result in a BadTiming error. If, during a FeatureDrawRequest, a call to this routine is NOT made, then the Draw Module will unconditionally send a FeatureNotAvailable error back to the Chart Manager.

ERRORS

BadServer

An invalid server id was used.

BadTiming

This routine has been called at an inappropriate time. GenChangedFeature() should be called only after a FeatureDrawRequest has been received.

SEE ALSO

GenAttach(3Gen), GenFeatDraw(3Gen), GenRequest(3Gen), MAddFeature(3Map)

GenChangedMap

FUNCTION

Notify Chart Manager that the Draw Module has completed the rendering of a map.

SYNTAX

C Interface

```
void GenChangedMap( server)
        ServerId    server;
```

ARGUMENTS

server The link between the Draw Module and the Chart Manager to which it is connected. Returned by GenAttach().

DESCRIPTION

The GenChangedMap() is used by a Draw Module to signify that a map which it is currently rendering has completed. This information is needed by the Chart Manager so that it can coordinate the map draw requests, and handle the completion of a client draw request. This call should be sent by a Draw Module ONLY IF the module has previously received a MapDrawRequest, and the draw module has been working on the rendering of the map request. Because of this, the implied parameters in the call reference the draw request currently being worked on.

When a Draw Module receives a MapDrawRequest, the module can either choose to ignore it, or redraw the map. If the latter choice is taken, the draw module which is rendering the map should send periodic updates to the Chart Manager as to the progress being made using the GenDrawingMap() call. The Chart Manager must then be notified when the rendering completes using the GenChangedMap() call. Calls to this routine at other times result in an error. If during a Map- DrawRequest a call to this routine is NOT made, then the Draw Module will unconditionally send a MapNotFound error back to the Chart Manager.

Draw Modules which are rendering a feature use the similar calling sequence of GenDrawingFeature() and GenChanged- Feature().

ERRORS

BadServer

An invalid server id was used.

BadTiming

This routine has been called at an inappropriate time. GenChangedMap() should be called only after a Map- DrawRequest has been received.

SEE ALSO

GenAttach(3Gen), GenChangedFeature(3Gen), GenMapDraw(3Gen), GenRequest(3Gen), MChangeMap(3Map)

GenClip

FUNCTION

Line/Polygon clipping and conversion routines.

SYNTAX

C Interface

```
MapStatus FormLine ( conv, pt1, pt2, gcrl,
    set )
ProjectionData  *conv;
MapPoint        *pt1;
MapPoint        *pt2;
int     gcrl;
PointSet        *set;

MapStatus AppendLine ( conv, pt1, pt2, gcrl,
    set )
ProjectionData  *conv;
MapPoint        *pt1;
MapPoint        *pt2;
int     gcrl;
PointSet        *set;

MapStatus FormPolyLine ( conv, pts, npts, gcrl,
    set )
ProjectionData  *conv;
MapPoint        *pts;
int    npts;
int     gcrl;
PointSet        *set;

MapStatus AppendPolyLine ( conv, pts, npts, gcrl,
    set )
ProjectionData  *conv;
MapPoint        *pts;
int    npts;
int     gcrl;
PointSet        *set;

MapStatus FormPolygon ( conv, pts, npts, gcrl,
    set )
ProjectionData  *conv;
MapPoint        *pts;
int    npts;
int     gcrl;
```

```
        PointSet                *set;

    MapStatus AppendPolygon ( conv, pts, npts, gcrl,
            set )
        ProjectionData  *conv;
        MapPoint                *pts;
        int    npts;
        int        gcrl;
        PointSet                *set;

    void InitializePointSet ( set )
        PointSet *set;
    void ClearPointSet (  set )
        PointSet *set;

    void FreePointSet (  set )
        PointSet *set;

    int SetPointInterpolation ( value )
        int value;
```

ARGUMENTS

conv	A projection data structure. Usually the structure which is returned by a MapDrawRequest or FeatureDrawRequest.
pt1	A Geodetic point on the world. Value is in radians.
pt2	A Geodetic point on the world. Value is in radians.
pts	A list of geodetic points on the world. All values are in radians.
npts	The number of points in the list pts.
gcrl	Line description type. Two values are supported: RhumbLine will connect line segments using bearing/range. GreatCircle will connect line segments using great circles.
set	A PointSet structure. This structure contains one or more lists of pixel points which describe the geometric entity after calculating, interpolating, clipping, and converting.

DESCRIPTION

Chart provides a number of routines to aid Draw Modules in rendering polygons, polylines, and lines which are geographically described using geodetic coordinates.

All of these routines require some sort of geodetic input coordinates, described using one or more MapPoint structures, a coordinate conversion structure, described by the ProjectionData structure, a line type value (usually set to RhumbLine, however GreatCircle is also supported), and an initialized PointSet structure, described below.

The most basic of the routines is FormLine(), which connects two geodetic points with an interpolated set of points. The

returned PointSet structure contains one or more arrays of XPoint records, which can be used as input to XDrawLines(3X11). The returned set of line segments is interpolated based on the following criterea:

(1). The line connection algorithm (rhumbline or great circle).
(2). The projection (some projections project lines as curves
(3). The current interpolation value as specified in the last call to SetPointInterpolation().

The geodetic line will be broken into more than one set of line segments if a geodetic boundary is crossed. Boundary crossing are projection dependent. The returned PointSet contains information as to the number of line segment sets are calculated.

Contiguous geodetic line segments can be connected by calling FormLine(), followed by successive calls to Append- Line(). A similar functionality exists for connecting multiple geodetic line segments by using the FormPolyLine() routine. The AppendPolyLine() routine acts like FormPoly- Line(). However, the output line segments are appended to the input PointSet.

The FormPolygon() routine calculates one or more polygons which will render the input polygon describe by the geodetic set of points in the pts list. The first and last points in this list are automatically closed if the two point's values are not equal. Polygons which cross viewing boundaries will be broken up, and connected as necessary. The AppendPo- lygon() acts like FormPolygon(), but the output points are appended to the current PointSet.

The Form...() routines will automatically clear up the PointSet structure prior to calculating the output segments using the ClearPointSet() call. The Append...() routines, on the other hand, do not clear the PointSet structure.

The call ClearPointSet() reinitializes a PointSet set structure WITHOUT freeing up already allocated memory. This is a more efficient approach when numerous calculations are being performed (such as drawing a lot of polygons in VectorDraw), because memory doesn't need to be constantly reallocated and freed.

FreePointSet() should always be called when the Draw Module no longer has a need for the information in the PointSet. This both clears the PointSet structure and frees any memory which it has previously allocated.

InitializePointSet() MUST be called prior to using a PointSet in any other routine! This is extremely important. Otherwise, unexpected results can occur.

SetPointInterpolation() can be called at any time to set a new interpolation value. The default value is currently 20, which means that as many as 20 points will be calculated for EACH input geodetic segment. If your draw module is already doing interpolation then set this value to 0.

Points which are so close together that their pixel equivalent values are the same are thrown out prior to conversion. This is dependent of course on the current viewing scale.

RETURN

The Form...() and Append...() routines return a value of NoError if the input values are valid and if the output PointSet is successfully created. An error value as described in the ERRORS section will be returned otherwise.

The SetPointInterpolation() routine returns the previous interpolation value. This allows the caller to set the interpolation back to the previous value when done.

STRUCTURES

C Interface

```
typedef struct _MapPoint {
    FLOAT   lat;
    FLOAT   lon;
    FLOAT   alt;
} MapPoint;

typedef struct {
        XPoint *points;
        short npoints;
        Boolean validpt;
} IntPointSet;
                /*Note: ignore other fields in this structure*/

typedef struct {
        IntPointSet *sets;
        short nsets;
} PointSet;
/*Note:  ignore other fields in this structure*/
```

The PointSet structure describes the set of points which will render the given geometric figure. The fields in this structure are as follows:

sets

A set of IntPointSet structures. Each of these structures contains a set of (x,y) pixel values for rendering part of the geometric figure. The IntPointSet structure is described in more detail below.

nsets

The number of sets described above. This specifies the MAXIMUM number of pixel sets which will describe the geometric figure. Note that each set must still be checked for validity prior to rendering.

The IntPointSet structure describes a set of pixels used for rendering at least part of the geometric figure. The fields in this structure which are of use to Draw Modules are as follows:

points

A set of pixel values which can be used in a XDrawLines(3X11) or XFillPolygon(3X11) call, as appropriate.

npoints

The number of points specified in points. Draw Modules should check to insure that this is a reasonable value (eg. greater than 1).

validpt

This value will be set to False if the pixel values are invalid, True otherwise. Draw Modules should check to see that this value is True prior to using the points for rendering.

ERRORS

Errors are returned in a MapStatus integer. Valid errors include:

BadValueError

An invalid point was specified, or else the npts field is unreasonable (less than or equal to 1).

OutOfMemory

Unable to allocate sufficient memory to perform calculations.

PointsTooClose

May be returned by FormLine() or AppendLine(). Indicates that no points were produced because the too points are sufficiently close to be treated as simply

one point on the given view. This provides dynamic clipping of lines, polygons, etc.

NOTES

(1). All geodetic coordinates are in radians, not degrees.

(2). When using XFillPolygon(3X11), it is probably better to specify Complex.

(3). The mode to use for either XFillPolygon(3X11) or

XDrawLines(3X11) should be CoordModeOrigin.

SEE ALSO

GenCoord(3Gen), GenFeatDraw(3Gen), GenMapDraw(3Gen),
MGetProjectionData(3Map), MPositionToPixels(3C), MPixelsToPosition(3C)
XDrawLines(3X11), XFillPolygon(3X11)

GenCoord

FUNCTION

Coordinate conversion routines.

SYNTAX

C Interface

```
Boolean PositionToPixels ( conv, lon, lat, x, y ) ProjectionData *conv;
        FLOAT          *lon;
        FLOAT          *lat;
        short          **x;
        short          **y;

Boolean PixelsToPosition ( conv, x, y, lon, lat ) ProjectionData *conv;
          int          x;
          int          y;
          FLOAT        *lon;
          FLOAT        *lat;

Boolean ComputeScales ( conv, lon1, lat1, p1 x, p1 y, lon2, lat2,p2 x, p2 y  )
        ProjectionData *conv;
        FLOAT lon1;
        FLOAT lat1;
        FLOAT   p1 x;
        FLOAT   p1 y;
        FLOAT lon2;
        FLOAT lat2;
        FLOAT   p2 x;
        FLOAT   p2 x;
```

ARGUMENTS

conv	A pointer to a ProjectionData structure. Usually the structure which is returned by a MapDrawRequest or FeatureDrawRequest. As an input to ComputeScales(), certain elements in the conv record must be filled in prior to calling ComputeScales(). Upon return, conv contains the filled in values necessary for conversion between geodetic and pixel coordinate systems. The returned structure is used only as an input by the PositionToPixel(), and PixelToPosition() calls to accompilish this.
lon lat	Longitude and latitude (in radians) of a point.
x y	X- and y- coordinate (in window coordinates) of a point.
lon1	

lat1 — The 1st cross reference point (longitude/latitude) for scaling, in radians.

p1 x

p1 y — The 1st cross reference point (x coordinate, y coordinate), for scaling, in window coordinates.

lon2

lat2 — The 2nd cross reference point (longitude/latitude) for scaling, in radians.

p2 x

p2 y — The 2nd cross reference point (x coordinate, y coordinate), for scaling, in window coordinates.

DESCRIPTION

The PositionToPixels() function converts a geodetic point on the geographic display to its pixel value. If the input value is not visible on the geographic display image, then False is returned; otherwise True is returned.

The PixelsToPosition() function converts a pixel point on the geographic display to its geodetic value. If the specified pixel position does not correspond to a point on the viewable map surface, then False is returned; otherwise True is returned. The current projection and scale are taken into account when converting the pixel location to a geodetic coordinate. The pixel locations are mapped to the geographic image starting from the upper left hand corner of the image. The point (0,0) represents the upper left hand corner of the pixmap, and (width, height) represents the lower right hand corner. Negative pixel values represent points above and to the left of the upper left hand corner, and may be valid provided that the viewable map space extends beyond the corners of the window.

The ComputeScales() function calculates the scaling factors necessary to perform future conversions between geodetic points and pixel points. Normal Draw Modules and Chart Clients do NOT need to use this function. Chart Clients can instead use the MPositionToPixels(3C) and MPixelsToPosition(3C) calls.

Pixel coordinate systems are rectangular grids of integer values. A unit of measure in either direction corresponds to so much change in latitude and/or longitude. The unit change is almost always non-linear, depending on both the

projection and earth model currently in use.
The ProjectionData structure is a structure used for converting between the geodetic and pixel coordinate systems. Certain fields in this structure must be filled in prior to calling ComputeScales().

RETURN

For the PositionToPixels() routine, the value True is returned when a valid conversion takes place, and the geodetic coordinate is viewable. The value False is returned when either the input point is not valid, or else the conversion fails, or else the geodetic point is not visible on the geographic display. If the call fails, then the pixel points are set to 0 as a precaution.

For the PixelsToPosition() routine, the value True is returned when a valid conversion takes place, and the pixel location lies on the world. The value False is returned when the pixel location does not lie on the world, or is otherwise not convertible. In the latter case, the geodetic coordinate is set to infinity as a precaution.

For the ComputeScales() routine, the value True is returned if the call succeeds, and False is returned if the call fails. The ProjectionData structure should be used only if the call succeeds.

STRUCTURES

C Interface

```
typedef struct {
    Boolean whole_world;
    Boolean crosses_pole;
    ProjectionType projection;
    EarthModel system;
    FLOAT pixel_x_origin;
    FLOAT pixel_y_origin;
    int width;
    int height;
    FLOAT dc_x_origin;
    FLOAT dc_y_origin;
    FLOAT lng_origin;
    FLOAT lat_origin;
    FLOAT sp1;
    FLOAT sp2;
    FLOAT x_scale;
    FLOAT y_scale;
    FLOAT factor;
    FLOAT dc_x_max;
    FLOAT dc_y_max;
  } ProjectionData;
```

The ProjectionData structure describes the coordinate transformation between geodetic and pixel coordinate systems. Certain fields within this structure must be filled in PRIOR to calling ComputeScales(). Other fields are filled in by the ComputeScales() routine. The other fields in the ProjectionData structure are for internal use only. The fields which need to be filled in by the Draw Module or Chart Client are:

whole_world

Set this field to True only if the geodetic coordinate system covers the entire world. Set this field to False otherwise.

projection

Set this field to the projection being used in the geodetic coordinate system. Valid projection values are found on the MProjection(3Map) man page.

system

Set this field to the Earth Model being used. This field must be set to one of the following values: Spherical or Elliptical.

width

Set this field to be the width of the pixel coordinate system. For example, the Window coordinate system sets this value to the window width. The pixmap coordinate system sets this value to the pixmap width.

height

Set this field to be the height of the pixel coordinate system. For example, the Window coordinate system sets this value to the window height. The pixmap coordinate system sets this value to the pixmap height.

lng_origin

Set this field to the longitude point which corresponds to the center point of the pixel coordinate system.

lat_origin

Set this field to the latitude point which corresponds to the center point of the pixel coordinate system.

sp1 sp2

These values are the standard parallels being used within the particular projection. Depending on the projection, none, one, or both of these values are used in the projection calculations.

NOTES

(1). All geodetic coordinates are in radians, not degrees. (2). These routines are accessible to Chart Clients as well, but are documented here because their usage is more common to Draw Modules. They do not require any interaction with the Chart Manager.

SEE ALSO

GenFeatDraw(3Gen) GenMapDraw(3Gen), MGetProjectionData(3Map), MPositionToPixels(3Map), MPixelsToPosition(3Map)

GenDetach

FUNCTION

Close a communication channel to the Chart Manager.

SYNTAX

C Interface

```
void GenDetach( server )
        ServerId  server;
```

ARGUMENTS

server The connection to the Chart Manager; returned by GenAttach().

DESCRIPTION

GenDetach() closes the communication channel between the Draw Module and the Chart Manager specified by server. The Chart Manager clears any products supported by this map generator client from the server's product list, and places them back on the unsupported product list.

ERRORS

BadServer

The server id is invalid.

SEE ALSO

GenAttach(3Gen), GenRemoveProducts(3Gen)

GenDrawingFeature

FUNCTION

Notify Chart Manager that a feature is being drawn.

SYNTAX

C Interface

```
void GenDrawingFeature( server, percent drawn, abort ) ServerId   server;
                                                       int        percent drawn;
                                                       Boolean    *abort;
```

ARGUMENTS

server The link between the Draw Module and the Chart Manager to which it is connected. Returned by GenAttach().

percent drawn

The percentage of the feature which has been drawn. This value should lie between 0 and 100.

abort

An output flag. It will be set to True if the current draw request should be aborted. It will be set to False otherwise.

DESCRIPTION

The GenDrawingFeature() call notifies the Chart Manager that the FeatureDrawRequest just sent by the Chart Manager to this Draw Module is being worked on. Multiple calls to this routine should be made to indicate progress while a feature is being rendered. In fact, Draw Modules should periodically make calls to this routine for feature products which take a while to draw. Generally speaking, a minimum of one call per every 5 seconds should be attempted. Calls to Gen- DrawingFeature() are valid ONLY IF the Draw Module is currently processing a FeatureDrawRequest. Calls to this routine at other times result in a BadTiming error.

A Draw Module which calls GenDrawingFeature() should call GenChangedFeature() once the feature draw is complete. Failure to do so will cause the module library to automatically send an ErrorDrawingFeature error to the Chart Manager, which aborts rendering of this feature.

A call to GenDrawingFeature() automatically checks the pending request queue to see if any abort requests have been received. If so, the calling module is notified via the abort parameter that the current draw request should be terminated.

Note: failure to send a draw response within the system specified time of COACH_MONITOR_INTERVAL will cause the

request to be terminated. This is to prevent draw modules from hanging up the system.

COACH_MONITOR_INTERVAL is a system parameter that is at least 5 seconds.

ERRORS

BadServer

An invalid server id was used.

BadTiming

This routine has been called at an inappropriate time. GenDrawingFeature() should be called only after a FeatureDrawRequest has been received.

SEE ALSO

GenAttach(3Gen), GenChangedFeature(3Gen), GenFeatDraw(3Gen), GenRequest(3Gen)

GenDrawingMap

FUNCTION

Notify Chart Manager that a map is being drawn.

SYNTAX

C Interface

```
void GenDrawingMap( server, percent drawn, abort ) ServerId    server;
                                                   int         percent drawn;
                                                   Boolean     *abort;
```

ARGUMENTS

server The link between the Draw Module and the Chart Manager to which it is connected. Returned by GenAttach().

percent drawn

The percentage of the map which has been drawn. This value should lie between 0 and 100.

abort An output flag. It will be set to True if the current draw request should be aborted. It will be set to False otherwise.

DESCRIPTION

The GenDrawingMap() call notifies the Chart Manager that the MapDrawRequest just sent by the Chart Manager to this Draw Module is being worked on. Multiple calls to this routine can be made to indicate progress as a map is being drawn. In fact, Draw Modules should periodically make calls to this routine for map products which take a while to draw. Generally speaking, a minimum of one call per every 5 seconds should be attempted. Calls to GenDrawingMap() are valid ONLY IF the Draw Module is currently processing a MapDrawRe- quest. Calls to this routine at other times result in a BadTiming error.

A Draw Module which calls GenDrawingMap() should call Gen- ChangedMap() once the map draw is complete. Failure to do so will cause the module library to automatically send a ErrorDrawingMap error to the Chart Manager, which aborts rendering of this map image.

A call to GenDrawingMap() automatically checks the pending request queue to see if any abort requests have been received. If so, the calling module is notified via the abort parameter that the current draw request should be terminated.

Note: failure to send a draw response within the system specified time of COACH_MONITOR_INTERVAL will cause the request to be terminated. This is to prevent draw modules

from hanging up the system. COACH_MONITOR_INTERVAL is a

system parameter that is at least 5 seconds.

ERRORS

BadServer

An invalid server id was used.

BadTiming

This routine has been called at an inappropriate time. GenDrawingMap() should be called only after a Map- DrawRequest has been received.

SEE ALSO

GenAttach(3Gen), GenChangedMap(3Gen), GenMapDraw(3Gen), GenRequest(3Gen)

GenError

FUNCTION

Draw Module error handling routines.

SYNTAX

C Interface

```
#include <M/Generror.h>
#include <M/Genproto.h>

void GenSetErrorHandler( handler)

                    GenErrorProc    handler;

                  void GenResetErrorHandler( )

void GenSetIOErrorHandler( handler) GenIOErrorProc    handler;

                  void GenResetIOErrorHandler( )

char *GenErrorToString( error code) MapStatus    error code;

                    char *GenRequestCodeToString( code)
              GenProtocol  code;
```

ARGUMENTS

handler — An application-specific error handler.

error code — The code number of the generated error. Error codes are described in each Draw Module manual page under the "ERRORS" heading.

DESCRIPTION

The Gen Library has two asynchronous error handler routines that are called whenever an error occurs. One handler deals exclusively with IO errors and the other handles all other errors. The Gen library's default error handler prints a message to the standard error device.

Both error handlers can be replaced by user defined handlers by using the routines GenSetIOErrorHandler() and GenSetEr- rorHandler(). Once the error handler is replaced by a user routine, this routine will be called whenever an error occurs. The default handlers that the library defines can be restored with the routines: GenResetIOErrorHandler() and GenResetErrorHandler().

Available to the user's error handlers are two routines for converting internal Chart Manager codes to strings. These routines are: GenErrorToString(), and GenRequestCodeTo- String().

The GenErrorToString() routine provides a small text description for each error code, and the GenRequestCodeTo- String() routine provides a text description for each Draw Module request code in the Chart Manager. Both routines return the text string "Unknown" in the case where the input code is not defined by the Chart Manager. Both routines return pointers to static string buffers which should not be modified by the caller.

The application error handler routine is called with the following format whenever a Chart Manager error occurs:

(*handler) (server, request_code, error_code)
    ServerId server;
    GenProtocol request code;

    MapStatus error code;

The IO error handler routine is called with the following format whenever an IO error occurs:

(*handler) (server, error_code)

    ServerId server;
    MapStatus error code;

The error code passed to the IO Error Handler will most likely be SocketError. This is a general error indicating that an error has occurred while trying to communicate over the socket. If more information can be obtained, a more specific error code will be returned.

The server indicates the server over which the error occurred. NOTE: Calls to routines which do not communicate with the Chart Manager are defined to return a status immediately.

Other calls, including those documented in MMemory(3Map) and MuReference(3Mu), and which might be used by Draw Modules, invoke separate error handling. See MError(3C) for details on this error handling.

The request code can be used by the Draw Module to handle failures due to a particular request. The error code parameter can be used by the Draw Module to handle specific errors.

The MResetErrorHandler() call resets the error handler back to the default error handler.

RETURN

The routines GenErrorToString() and GenRequestCodeToString() return a string value. This value is statically assigned , and should NOT need be freed using MFree.

STRUCTURES

C Interface

```
typedef int MapStatus;
typedef short GenProtocol;
typedef void (*GenErrorProc)(

        ServerId   server, GenProtocol message_type, MapStatus  error_code);

typedef void (*GenIOErrorProc)( ServerId   server, MapStatus  error_code);
```

VALUES

MapStatus

AllocationFailure

Unable to allocate indicate resource. Specifically occurs on allocation of colors.

AlreadyConnected

A connection to this Chart Manager already exists. Only one connection to any one Chart Manager is allowed.

BadDisplay

The indicated GenDisplay does not exist, or else has been deleted.

BadServer

The indicated ServerId is invalid.

BadTiming

The indicated response is inappropriate under the current circumstances.

BadValueError

An input parameter to one of the Gen library calls is invalid.

BadWindow

The indicated geographic display window does not exist, or else has been deleted.

DataSyncError

The Draw Module/Chart Manager protocol is out of sync. Some data may be lost.

DisplayOpen

Unable to open a screen display within this Draw Module. This may occur when a Draw Module tries to open another host's display, yet does not have privilege.

FeatureAlreadyClaimed

The specified feature has already been reserved by another Draw Module. See GenReserveFeature().

MapAlreadyClaimed
The specified map has already been reserved by another Draw Module. See GenReserveMap().

PointsTooClose
Used in certain GenClip(3Gen) calls for returning clipping information to the caller.

OutofMemory
The system's swap storage is full and is unable to allocate additional memory.

SocketError
An exception occurred on one of the Gen library's communcation sockets. This error code is generally associated with the I/O error handler.

UnknownError
The specified problem is undefined.

SEE ALSO

Gen-Intro(3Gen), MError(3C), MMemory(3Map), MuReference(3Mu),ERRORS section under each application call.

GenFlushAllRequests

FUNCTION

Flush the Draw Module request queue.

SYNTAX

C Interface

```
void GenFlushAllRequests();
```

DESCRIPTION

The GenFlushAllRequests() routine clears the Draw Module request queue. Each pending request is first looked at, and if any action is required, the default action is taken.

SEE ALSO

GenRequest(3Gen)

GenGetDisplay

FUNCTION

Get X Window display record from the Gen-Display code.

SYNTAX

C Interface

```
#include <X/Xlib.h>
Display *GenGetDisplay( ndpy )
        int  ndpy;
GenDisplay _GetGenDisplay ( map window )
        WindowId  map window;
```

ARGUMENTS

ndpy — The GenDisplay identifier, which is a field present in many GenRequest structures, and which serves as a reference to the X Window screen/display.

map window — The geographic window identifier.

DESCRIPTION

The GenGetDisplay() routine returns an X window display record, given an int code as an input. The GetGenDisplay() routine returns a GenDisplay code, given the geographic window as an input.

RETURN

The GenGetDisplay() routine returns a pointer to an X windows Display structure. The value NULL is returned on error.

The _GetGenDisplay() routine returns the Gen library's display identifier for a given geographic window. The value -1 is returned on error.

STRUCTURES

C Interface

```
typedef int GenDisplay;
```

The GenDisplay construct is used by the Gen library to reference the display hardware. The Gen library caches references to the display hardware to minimize the number of XOpenDisplay(3X11) calls made by a draw module.

ERRORS

BadDisplay

The specified ndpy is an invalid or non-existent display.

BadWindowId

The specified map window is invalid or non-existent.

SEE ALSO

XOpenDisplay(3X11)

GenInit

FUNCTION

Draw Module Standardized Initialization Functions.

SYNTAX

C Interface

```
#include <M/Genlib.h>
#include <M/GenInit.h>

int GenInitialize( argc, argv, inatts, inmask, outatts) int   argc;     /*Input*/
      char  **argv;   /*Input*/ DrawModuleInputAttributes *inatts;  /*Input*/
      MapValueMask    inmask;   /*Input*/ DrawModuleOutputAttributes
      *outatts; /*Output*/
```

ARGUMENTS

argc — A count of the number of arguments passed to the Draw Module when the command is invoked.

argv — The NULL terminated list of command line arguments. See execv(3) for more information.

inatts — A structure which describes special initialization attributes over and above the default attributes used to initialize the Draw Modules's connection to the Chart Manager. The fields in this structure are discussed further in the STRUCTURES section of this manual page.

inmask — A bit mask which indicates those fields in inatts which are being specified. Those fields not specified imply certain default actions. The bit mask values are described in the BIT MASK section of this manual page.

outatts — A structure containing key fields and values for communicating with the Chart Manager, and to a connected window. The fields are discussed further in the STRUCTURES section of this manual page.

DESCRIPTION

Draw Modules are encouraged to use the GenInitialize() function to provide a consistent interface to the Chart Manager and to various Chart Windows from disparate Draw Modules. GenInitialize() provides the following services to a Draw Module:

(1). Connection to Chart Manager.

(2). Consistent and proper error handling in the event

the connection fails.
(3). Consistent handling of a core set of command line options. The ability to parse additional options special to a particular Draw Module is also supported.

When GenInitialize() is invoked the following actions take place:

(1). The command line arguments are parsed against the core options and any additional options specified by your Draw Module. Syntax errors and/or inconsistent handling of the command line arguments results in an error message being printed to the standard error device, the command's proper syntax being printed out, and the program returning -1. GenInitialize() makes use of the MuOption(3Mu) routines to parse the command line options. See OPTIONS section of this manual page for a list of core options supported.

(2). A connection to a Chart Manager is attempted. Generally this connection takes place with the Chart Manager on the same machine, unless a command line option specifies otherwise. If the connection fails, repeated attempts may be made to connect to Chart Manager after a certain delay period. This depends on the options specified on the command line. Eventually if no connection occurs, an error message is printed to the standard error device, and the program returns -1.

RETURN

GenInitialize() returns -1 if any problems occur, and returns 0 otherwise.

STRUCTURES

C Interface

```
typedef struct _DrawModuleInputAttributes{
    Qualifiers *options;
} DrawModuleInputAttributes;

typedef struct _DrawModuleOutputAttributes {
    char  hostname[64];
    char **extra_values;
    Boolean *present;
    ServerId server;
} DrawModuleOutputAttributes;
```

The DrawModuleInputAttributes structure provides GenInitial- ize() with additional information on how the Draw Module is to be initialized. Fields within this structure are checked

only if the corresponding bit in the inmask bit mask is set. The fields in this structure are described as follows:

options
: A set of additional command line options which this client supports. Refer to the MuOption(3Mu) manual page for information on filling in the fields of the Qualifiers structure. Note that this list is NULL terminated. If no options are specified, then only the core set of options are validated.

The DrawModuleOutputAttributes structure provides the Draw Module with the information needed to communicate with the Chart Manager. The fields in this structure are described as follows:

hostname
: The name of the machine where Chart Manager is running, and to which this Draw Module has connected. This is usually the host name of the machine where the Draw Module is running, but the "-host" option can be used to specify an alternate host.

extra_values
: A NULL terminated list of command line parameters which are not options. These are available to the Draw Module for processing.

present
: A list of Boolean flags which indicate whether or not the extra options which a Draw Module has specified are present on the command line or not. The list is indexed in an identical manner to the list of Qualif- iers in the DrawModuleInputAttributes structure. A value of True for one of the members in the list indicates that the corresponding option has been specified. A value of False indicates that it is absent from the command line.

  If no extra options have been specified, then this list will be set to NULL. C programmers should free this space using a call to MFree when it is no longer needed.

server
: The internal connection identifier used to communicate with the Chart Manager. This value is needed as a parameter in virtually all Gen library routines.

OPTIONS

GenInitialize() recognizes the following set of core command line options for all Draw Modules which call it.

-delay time
: Specify a delay period to wait between retries. The time value is in seconds. Default value is 5 seconds.

-host host

Specify the name of the host the NTCS Chart Manager is running on. This is the Chart Manager to which the Client process will attempt to attach. The default host is the machine where this Draw Module is executing.

-help

Request help information concerning qualifier usage. This will print out the command syntax, and supported options for the given Draw Module, and return -1.

-retry [retries]

Specify that if a connection to the Chart Manager fails, then this Draw Module should periodically retry its connection to Chart Manager. The optional retries value specifies the number of retries to make before giving up. The "-delay" option is available to modify the period. The default number of retries, if none are specified, is 10. However, the default option if neither "-retries" nor "-delay" is specified is to simply return -1 if the connection fails.

-noretry

Specify that if a connection to Chart Manager fails, then the Draw Module simply returns -1. This is the default action.

BIT MASK

The inmask parameter is a bit mask which allows the Draw Module to selectively specify those input attributes which it has specified. The following values are allowed:

DMAll

All bits are set.

DMNone

No bits are set.

DMQualifiers

This bit indicates that the command syntax for this Draw Module, as specified in the qualifiers field, supports additional command line options besides those specified in the OPTIONS section of this manual page.

SEE ALSO

GenAttach(3Gen), MuInit(3Mu), MuOption(3Mu),

GenNextRequest

FUNCTION

Obtain next request/notify from Chart Manager.

SYNTAX

C Interface

```
void GenNextRequest( request )
        GenRequest *request;
```

ARGUMENTS

request The next request on the request queue.

DESCRIPTION

GenNextRequest() obtains the next request from the Chart Manager. If the request queue is empty, GenNextRequest() blocks until a request is received. GenNextRequest() is set up to receive events from more than one Chart Manager if the Draw Module is connected to more than one.

A returned request may require some actions by the Draw Module prior to another call to GenNextRequest(). For example, a GenMapDrawRequest requires the Draw Module to call the routines GenDrawingMap() and GenChangedMap() prior to receiving another request. Failure to do so will result in default actions taking place upon the next call to GenNex- tRequest() or GenPending().

ERRORS

OutOfMemory

Unable to allocate memory for request record.

SEE ALSO

GenAttach(3Gen), GenChangedFeature(3Gen), GenChangedMap(3Gen), GenDrawingFeature(3Gen), GenDrawingMap(3Gen), GenMapDraw(3Gen), GenFeatDraw(3Gen), GenPending(3Gen), GenRequest(3Gen)

GenPending

FUNCTION

Return number of pending requests.

SYNTAX

C Interface

int GenPending( )

DESCRIPTION

GenPending() returns the number of Chart Manager requests pending for this Draw Module. The total number of events for all Chart Manager connections is returned. The GenNex- tRequest() call can be used to retrieve the actual request. This call should not be made before the previous request has been serviced (see GenNextRequest()).

RETURN

The function returns the number of events still on the event queue. A value of 0 is returned if no events are presently on the queue.

SEE ALSO

GenNextRequest(3Gen), GenRequest(3Gen)

GenRemoveFeatures

FUNCTION

Remove features which have been of interest to this Draw Module.

SYNTAX

C Interface

```
void GenRemoveFeatures( server, products, numproducts ) ServerId       server;
                                FeatureProduct     *products;
                                  int              numproducts;
```

ARGUMENTS

server The link between the Draw Module and the Chart Manager to which it is connected. Returned by GenAttach().

products Describes a list of feature products which are no longer of interest to this Draw Module.

numproducts
The number of products described in products.

DESCRIPTION

The GenRemoveFeatures() call specifies features which this Draw Module is no longer responsible for rendering. Specified products in products were previously specified in one or more GenAddFeatures() calls. If the Chart Manager has this Draw Module responsible for rendering a particular feature, and if one of the items in the products list matches this feature, then the Chart Manager will move the feature onto the unsupported feature product list. Additionally, a GenFeatureVerifyRequest is again sent to each Draw Module which had earlier expressed an interest in this product using the GenAddFeatures() call. Another Draw Module can then claim this feature using GenReser- veFeature().

The features are specified using FeatureProduct records. The value AnyFeature serves as a wildcard and is supported with any of the fields in the FeatureProduct structure, including the FeatureType field. The FeatureType, and FeatureSubType fields are the same fields used in the Map- FeatureAttributes structure.

STRUCTURES

C Interface

```
typedef struct _FeatureProduct {
        FeatureType    feature_type;
        FeatureSubType  sub_type;
} FeatureProduct;
```

ERRORS

BadServer

An invalid server id was used.

BadValueError

An invalid product specification was sent to the Chart Manager. The specified value for FeatureType or FeatureSubType is not supported by the Chart Manager.

SEE ALSO

GenAddFeatures(3Gen), GenRemoveProducts(3Gen), GenAttach(3Gen), MuReference(3Mu)

GenRemoveProducts

FUNCTION

Remove map products which have been of interest to this Draw Module.

SYNTAX

C Interface

```
void GenRemoveProducts( server, products, numproducts ) ServerId    server;
                                MapProduct   *products;
                                 int         numproducts;
```

ARGUMENTS

server The link between the Draw Module and the Chart Manager to which it is connected. Returned by GenAttach().

products Describes a list of map products which are no longer of interest to this Draw Module.

numproducts

The number of products described in products.

DESCRIPTION

The GenRemoveProducts() call specifies map products which this Draw Module is no longer responsible for rendering. Specified products in products were previously specified in one or more GenAddProducts() calls. If the Chart Manager has this Draw Module responsible for rendering a particular feature, and if one of the items in the products list matches this feature, then the Chart Manager will move the map onto the unsupported map product list. Additionally, a GenMapVerifyRequest is again sent to each Draw Module which has earlier expressed an interest in this product using the GenAddProducts() call. Another Draw Module can then claim this map using GenReserveMap().

The map products are specified using MapProduct records. The value AnyMap serves as a wildcard and is supported with any of the fields in the MapProduct structure, including the MapType field. The MapType, and MapSubType fields are the same fields used in the MapChangeAttributes structure.

STRUCTURES

C Interface

```
typedef struct _MapProduct {
        MapType      map_type;
        MapSubType   sub_type;
} MapProduct;
```

ERRORS

BadServer

An invalid server id was used.

BadValueError

An invalid product specification was sent to the Chart Manager. The specified value for MapType or MapSubType is not supported by the Chart Manager.

SEE ALSO

GenAddProducts(3Gen), GenRemoveFeatures(3Gen), GenAttach(3Gen), MuReference(3Mu)

GenReserveFeature

FUNCTION

Request ownership of indicated feature product.

SYNTAX

C Interface

void GenReserveFeature(server, feature id, feature type, atts, atts mask)

ServerId server; FeatureId *feature id; FeatureType feature type; GenUpdateFeatureAttributes *atts; MapValueMask atts mask;

ARGUMENTS

server The link between the Draw Module and the Chart Manager to which it is connected. Returned by GenAttach().

feature id
A unique identifier for the feature entry to be reserved by this Draw Module. Passed to the Draw Module as part of a FeatureVerifyRequest.

feature type
An identification as to the class of the feature being reserved. Used by Chart Manager to speed up the search for the product of interest.

atts A list of attributes as modified by the Draw Module.

atts mask A mask representing the attributes from the list which the Draw Module actually is modifying. The GenUpdMask(3Gen) manual page provides a list of valid values for this mask.

DESCRIPTION

The GenReserveFeature() call tells the Chart Manager that this Draw Module wishes to be responsible for all draws concerning this feature product. All subsequent requests to the Chart Manager to draw this type of feature will result in a FeatureDrawRequest being sent to this Draw Module. A Gen- RemoveFeatures() call that includes this feature's product codes disables Chart Manager from sending any more FeatureDrawRequests for this product to this Draw Module.

The GenReserveFeature() call can be made many times for the same feature entry in order to update the modifiable parameter list. If another Draw Module has already reserved this feature, however, a FeatureAlreadyClaimed error will occur because at most one Draw

Module can be considered

responsible for drawing this feature product, even if many Draw Modules need to use this product while drawing another feature product (such as a Draw Module which dynamically merges two database types).

STRUCTURES

C Interface

```
typedef struct {
    FLOAT scale;
    FLOAT scale_lower;
    FLOAT scale_upper;
    int num_projections;
    ProjectionType supported_projections[NUM_PROJECTIONS];
    MapValueMask default_mask;
    ColorAllocationScheme allocate;
    ColorModel  color_model;
    FeatureAttributes defaults;
} GenUpdateFeatureAttributes;
```

The GenUpdateFeatureAttributes structure contains a set of fields which the Draw Module can use to modify part of a feature description entry (see GenFeatEntry(3Gen) for a full description of a feature description entry). The fields for this structure are defined as follows:

scale

The Draw Module should provide a modified scale factor (in nautical miles per pixel). This value is seen whenever a Chart Client uses the information from a MListFeatures(3Map) call. The scale value calculations must be consistent across all Draw Modules in order for the Chart Manager to be able to accurately "glue" together disparate features. A scale value of AnyScale is not recommended, even though the Chart Manager doesn't flag it as an error. This is because certain special Chart Clients designed to identify and call up feature products would have to estimate a good scale for displaying the feature.

scale_lower

The Draw Module should provide an absolute lower bound that it supports for drawing this feature product. The scale value is in nautical miles per pixel. This bound can be set to AnyScale, which means that the Draw Module supports the drawing of this product with no lower bounds on scale. The Draw Module is guaranteed to receive no draw requests for this product at scales

smaller than scale lower.

scale_upper

The Draw Module should provide an absolute upper bound that it supports for drawing this feature product. The scale value is in nautical miles per pixel. This bounds CAN be set to AnyScale, which means that the Draw Module supports the drawing of this product with no upper bounds on scale. The Draw Module is guaranteed to receive no draw requests for this product at scales larger than scale upper.

num_projections

The size of the supported projections list which gets returned. The list size can be no larger than NUM_PROJECTIONS, which at compile time equals the number of different projections supported by the Chart Manager.

supported_projections

The Draw Module should provide a list of projections that it supports for drawing this feature product. It is acceptable to place the wildcard projection value of AnyProjection at any point in the list. If a Chart Client requests that a feature be rendered in an unsupported projection, then a ProjectionNotSupported error is sent back to the client. See MProjection(3Map) for additional information.

allocate

The color allocation scheme used for rendering the feature. Three values are supported: AllocateReadOnly, AllocateShared, and AllocateReadWrite. The latter value is the default, and supports the intensity color model; however, this allocation scheme allocates precious color resources for private use. Color resources may be shared on a limited basis, as a function of the current color model in effect for the feature. If AllocateReadOnly is specified, then the color gets shared regardless. AllocateShared works similarly to AllocateReadWrite; however, all color resources are looked at in obtaining a color for the feature. If a color cell is available, then it is allocated for private use; however, if a color cell is not available, then it is allocated from the current pool of allocated colors, based on the closest matching color. Allo- cateShared is also a degraded color mode, for cases where attempts to allocate colors in AllocateReadWrite mode fail.

color_model

The extent of sharing employed by the Chart Manager,

when the allocate scheme is AllocateReadWrite or Allo- cateShared. Four color models are supported for features:

DoNotShareColors

This model allocates the color for exclusive use by this feature. It will not share the color with other users unless there are insufficient resources to allow for the exclusive allocation to take place. Use this model if the feature will be modifying the color value internally.

ShareColors

This model will attempt to share this color with any other feature that renders with the same color, and which has allocated the color in a similar fashion.

ShareColorsInClass

This model will attempt to share this color with any other feature in the same class which requires the same color, and which has allocated the color in a similar fashion. This is the default model.

ShareColorsInProduct

This model will attempt to share this color with similar feature products which use the same color, and which have allocated the color in a similar fashion.

default_mask

Those feature attributes which this draw module is modifying. These so called "default" attributes will take on their "system default" values if they are not specified here. The "system default" values are defined on the MFeatAtts(3Map) manual page.

defaults

The default rendering attributes which this feature should take on. The description of each feature rendering attribute is described in the MFeatAtts(3Map) man page. Draw Modules are responsible for providing a default set of feature attributes for rendering the feature. If these defaults are not provided, then system defaults for the particular feature attribute are used instead. Client requests to render a feature also use a feature mask to change or modify part of the default feature attributes structure. This mask makes modifications to the default feature attributes only for specific cases (such as a client requests that roads be drawn as dashed lines, rather than solid lines).

ERRORS

BadServer

An invalid server id was used.

BadValueError

An invalid feature identifier specification, or an invalid color model was sent to the Chart Manager.

FeatureAlreadyClaimed

The specified feature entry has already been claimed by another Draw Module.

SEE ALSO

GenAttach(3Gen), GenFeatEntry(3Gen), GenFeatVerify(3Gen), MChangeMap(3Map), MFeatAtts(3Map), MListFeatures(3Map), MProjection(3Map),

GenReserveMap

FUNCTION

Request rendering responsibility of indicated map product.

SYNTAX

C Interface

```
void GenReserveMap( server, map id, map type, atts, atts mask )
        ServerId        server;
        MapId           map id;
                MapType         map type;
        GenUpdateProductAttributes *atts; MapValueMask atts mask;
```

ARGUMENTS

server The link between the Draw Module and the Chart Manager to which it is connected. Returned by GenAttach().

map id A unique identifier for the map entry to be reserved by this Draw Module. Passed to the Draw Module as part of a MapVerifyRequest.

map type An identification as to the class of the map being reserved. Used by Chart Manager to speed up the search for the product of interest.

atts A list of attributes as modified by the Draw Module.

atts mask A mask representing the attributes from the list which the Draw Module actually is modifying. The GenUpdMask(3Gen) manual page provides a list of valid values for this mask.

DESCRIPTION

The GenReserveMap() call tells the Chart Manager that this Draw Module wishes to be responsible for all draws concerning this map product. All subsequent requests to the Chart Manager to draw this type of map will result in a MapDrawRe- quest being sent to this Draw Module. A GenRemoveProducts() call that includes this map's product code disables Chart Manager from sending to this Draw Module any more MapDrawRe- quests for this product.

The GenReserveMap() call can be made many times for the same map entry to update the modifiable parameter list. If another Draw Module has already reserved this map, however, a MapAlreadyClaimed error will occur. At most one Draw Module can be considered responsible for drawing this map product, even if many Draw Modules need to use this product while drawing another map product (such as a Draw Module

which dynamically merges two database types).

STRUCTURES

C Interface

```
typedef int ColorModel;
typedef struct {
    FLOAT scale;
    FLOAT scale_lower;
    FLOAT scale_upper;
    int num_projections;
    ProjectionType supported_projections[NUM_PROJECTIONS];
    ColorModel color_model;
        ColorAllocationScheme color_allocation;
    MapColor *color_list;
    int ncolors;
} GenUpdateProductAttributes;
```

The GenUpdateProductAttributes structure contains a set of fields which the Draw Module can use to modify part of a map description entry (see GenMapEntry(3Gen) for a full description of a map description entry). The fields for this structure are defined as follows:

scale

The Draw Module should provide a modified scale factor (in nautical miles per pixel). This value is seen whenever a Chart Client uses the information from a MListMaps(3Map) call. The scale value calculations must be consistent across all Draw Modules in order for the Chart Manager to be able to accurately "glue" together disparate maps. A scale value of AnyScale is not recommended, even though the Chart Manager doesn't flag it as an error. This is because certain special Chart Clients designed to identify and call up map products would have to estimate a good scale for displaying the map.

scale_lower

The Draw Module should provide an absolute lower bound which it supports for drawing this map product. The scale value is in nautical miles per pixel. This bound can be set to AnyScale, which means that the Draw Module supports the drawing of this product with no lower bound on scale. The Draw Module is guaranteed to receive no draw requests for this product at scales smaller than scale lower.

scale_upper

The Draw Module should provide an absolute upper bound that it supports for drawing this map product. The

scale value is in nautical miles per pixel. This bound can be set to AnyScale, which means that the Draw Module supports the drawing of this product with no upper bound on scale. The Draw Module is guaranteed to receive no draw requests for

this product at scales larger than scale upper.

num_projections

The size of the supported projections list which gets returned. The list size can be no larger than NUM_PROJECTIONS, which at compile time equals the number of different projections supported by the Chart Manager.

supported_projections

The Draw Module should provide a list of projections that it supports for drawing this map product. It is acceptable to place the wildcard projection value of AnyProjection at any point in the list. Multiple references to a projection indicate a preference for drawing the map product in that projection. Chart Clients which request a new geographic display with a projection field set to AnyProjection allow the Chart Manager to determine the best projection to use based on projection preferences for each of the draw requests which make up the request. See MChangeMap(3Map) and MProjection(3Map) for additional information.

color_model

This field identifies the color modeling scheme employed for drawing this map. The valid values for this field are as follows:

OneColorSetPerMap

When set to this value, the color map is unique for the given map. A color map must be supplied, indicating the number of colors needed to render the map product. Example: Compressed Aeronautical Charts (CACs) employ a unique color map for each chart.

OneColorSetPerMapProduct

When set to this value, the color map is unique across all maps in a given map product. A map product is uniquely defined by its map type and map subtype fields (see GenMapEntry(3Gen)). All instances for a given map product share the same color map in this case. Only the first map being reserved needs to supply a color map. All similar map products use the same color scheme for rendering. Example, SPOT satellite images, a type of IMAGMap employs a grey scaling to render the

image.

OneColorSetPerMapClass

When set to this value, the color map is unique across all maps in a given map class. A map class is uniquely defined by its map type field (see GenMapEntry(3Gen)). All map products for a given map class will share the same color map. Only the first map product in the class needs to supply a color map. All map products in the same class will use the same color scheme for rendering. Example: ARC Digitized Raster Graphics (ADRG) map products use a common color map.

UseFBModel

When set to this value, the color model used is based on the foreground and background color values currently in effect. These colors are set and modified by the MSetMapColors(3C) and MSetMapColorsByRGB(3C) library calls, as well as via any call which includes the MapColorAttributes structure (see MColor(3Map). When in this mode, the list and ncolors fields are ignored, because the color values are derived from library calls. Example: Vector map products such as World Vector Shoreline and World Database use this scheme.

color_allocation

Specifies the type of color allocation needed for rendering this map product. The following values are allowed:

AllocateReadWrite

Specifies that the color map should be modifiable. Color maps allocated in this manner support the intensity model, which is modified by the MSetIntensity(3Map) library call.

AllocateReadOnly

Specifies that the color map is non-modifiable. Color maps allocated in this manner do not support the intensity model, nor can they be changed by any of the other color models (such as UseFBMo- del). However, colors allocated in this manner may be shared among many Chart Clients and Draw Modules.

AllocateShared

Specifies that the color map is modifiable to a limited degree. The color will be shared with other maps and features which allocate the same color. This color follows the intensity color

model, but should otherwise be treated as a nonmodifiable color. AllocateShared is used as a first choice degraded mode if the Chart Manager runs out of colors when allocating maps. This allows intensity to be supported, but to a more limited degree than AllocateReadOnly.

AllocateTrueColor

Specifies that no color map is provided. Furthermore, the Draw Module can render this map only on hardware which supports TrueColor color maps. A Draw Module may choose to render maps using a number of different color mapping schemes. In this case, the Draw Module should select one of the other two allocation schemes. When a Map- DrawRequest is subsequently received, the ColorAt- tributes can then be referenced to see if TrueColor is supported.

color_list

Specifies the list of colors required for rendering the map. Values are specified as triples of red, green and blue. Where intensity is supported, the Draw Module supplies the full intensity values here. Values of red, green, and blue must follow the XColor(3X11) model and should lie between 0 and 255. If the allocation scheme which is specified is AllocateTrueColor, then a color map should not be specified. The Draw Module is then responsible for the pixel values which are written to the display hardware. See MColor(3Map) for information about the MapColor structure.

ncolors

The number of colors in the supplied color map.

ERRORS

BadServer

An invalid server id was used.

BadValueError

An invalid map identifier specification, or an invalid color model was sent to Chart Manager. Note: if no color model is specified, Chart Manager will try and match this map product to an existing color model from a previous map product. If no match occurs, Bad- ValueError is returned.

MapAlreadyClaimed

The specified map entry has already been claimed by another Draw Module.

NOTES

(1). When colors are allocated AllocateReadWrite, limited color resources, and/or the resident display hardware, may result in the colors actually be allocated AllocateShared (first choice), or AllocateReadOnly. The draw request provide information on the color allocation which actually occurs. (2). Colors allocated AllocateReadWrite may be modified by the Draw Module, if desired. However, conflicts can arise when in degraded mode, and another map allocates some of these colors as AllocateShared.

SEE ALSO

GenAttach(3Gen), GenMapEntry(3Gen), GenMapVerify(3Gen), MChangeMap(3Map), MColor(3Map), MListMaps(3Map), MProjection(3Map), MSetIntensity(3Map), MSetMapColors(3C), MSetMapColorsByRGB(3C),

GenSendError

FUNCTION

Send an error to the Chart Manager.

SYNTAX

C Interface

```
void GenSendError( server, error code )
        ServerId    server;
            MapStatus    error code;
```

ARGUMENTS

server The link between the Draw Module and the Chart Manager to which it is connected. Returned by GenAttach().

error code

The error code to be sent. Valid errors which can be sent are described under VALUES.

DESCRIPTION

The GenSendError() call indicates to the Chart Manager that an error has occurred in the indicated map window. Errors sent from Draw Modules are routed into the Chart Manager's error handling mechanisms, and may get sent to individual Chart Clients. Some errors get sent by the Draw Module library automatically if the Draw Module fails to perform some required actions (see GenMapDraw(3Gen), for example). The GenSendError() routine should be called only if something unexpected happens. For example, if while drawing a map the network suddenly goes down, an ErrorDrawingMap call might be appropriate. But if the GenDrawingMap() call has already been made, then GenSendError() needs to be called to abort the draw.

VALUES

The following are legal values for error code:

BadMapEntry

An error occurred while trying to draw the requested map or feature. This error associates the cause with bad data formats, missing files, or other related problems.

ErrorDrawingMap

An error occurred while trying to draw a requested map. This is a catch-all error code for handling problems with a draw map request.

ErrorDrawingFeature

An error occurred while trying to draw a requested feature. This is a catch-all error code for handling problems with a draw feature request.

FeatureNotAvailable
The specified feature is unavailable.

MapDrawAborted
The specified map draw was aborted due at the request of the Chart Manager.

OutOfMemory
The Draw Module is unable to allocate enough memory.

ProjectionNotSupported
The Draw Module does not support this projection.

SystemNotSupported
The Draw Module does not support this earth model.

UnresponsiveDrawModule
Suggested error to send if Draw Module would core dump. Note that Draw Modules can trap certain errors using Unix signal handling. See signal(3) for more information.

ERRORS

BadServer
An invalid server id was used.

BadTiming
An attempt to send an error to the Chart Manager occurred at an inappropriate time. This routine should be called ONLY while servicing a draw request.

BadValueError
An invalid error code was specified.

BadWindowId
The specified map window is invalid, does not exist, or has been deleted.

SEE ALSO

GenAttach(3Gen), GenMapDraw(3Gen), GenError(3Gen), GenMapVerify(3Gen), MError(3C)

GenServerToSocket

FUNCTION

Return the file descriptor of a Chart Manager's socket connection.

SYNTAX

C Interface

```
int GenServerToSocket( server )
        ServerId  server;
```

ARGUMENTS

server The connection to the Chart Manager; returned by GenAttach().

DESCRIPTION

The GenServerToSocket() function returns the file descriptor of the connection to a Chart Manager, or (-1) if the specified Chart Manager is invalid. This function is useful if a Draw Module application is to make a call to select(2). This call is provided if a Draw Module needs to use the file descriptor, but is not recommended. Requests, errors, and responses should be handled through the routines provided in the Gen library.

RETURN

The file descriptor for the socket is returned. The value -1 is returned in the case of an error.

ERRORS

BadServer

The Server id was invalid.

SEE ALSO

GenAttach(3Gen), GenDetach(3Gen), intro(2), select(2)

JMS_ConfigAOIGet

1.0 FUNCTION: Retrieve the current AOI setting for the database connection.

2.0 DESCRIPTION: This function will access the database connection that was established through a previous call to JMS_DbConnect and retrieve the AOI. The default AOI is the AOI of the database the application is connected to. The AOI can be modified with a call to JMS_ConfigAOISet.

This function will allocate memory for the AOI. Therefore the JMTK application is responsible for freeing the memory when it is no longer needed.

3.0 SYNTAX:

#include "JMS.h"

#include "JMS_Errors.h"

tJmsStatus JMS_ConfigAOIGet(tJmsConnection *conn_id* , tJmsArea ***pAoi*)

4.0 ARGUMENTS:

conn_id	Used to identify the unique connection to the SDB.
pAoi	The address of a structure pointer of the type tJmsArea.

5.0 RETURNS: A status of type tJmsStatus is returned, as defined below. The return status will equal JMS_OPERATION_OK when the AOI has been successfully retrieved, and returned in the *pAoi* structure.

6.0 DEPENDENCIES: TBD

7.0 ERROR MESSAGES: The following messages are displayed when errors occur:

JMS_INVALID_CONNECT_ID:	Connection Id is not recognized
JMS_AOI_NOT_DEFINED:	The AOI for the connection has been set

8.0 OTHER APIs: See also JMS_ConfigAOISet and JMS_DbConnect.

9.0 RELATED DOCUMENTATION: Not applicable.

JMS_ConfigAOISet

1.0 FUNCTION: Define the geographic area of interest (AOI) of a spatial database connection.

2.0 DESCRIPTION: JMS_ConfigAOISet stores an area of interest that is associated with a specific database connection. The AOI that is stored is used as a filter for fulfilling database requests/services. This will limit the amount of data that needs to be processed by the database functions.

In order to move away from the former paradigms of rectangular AOIs, this function expects a list of geographic points of latitude/longitude pairs which are stored as decimal degrees in clockwise order. The minimum number of points is four.

3.0 SYNTAX:

#include "JMS.h"

#include "JMS_Errors.h"

tJmsStatus JMS_ConfigAOISet(tJmsConnection *conn_id* , tJmsArea **pAoi*);

4.0 ARGUMENTS:

conn_id	Used to identify the unique connection to a SDB.
pAoi	Holds the number of points and the latitude/longitude pairs for the points.

5.0 RETURNS: A status of type tJmsStatus is returned. The status will equal JMS_OPERATION_OK when the passed AOI bas been successfully set for the specified database connection.

6.0 DEPENDENCIES: TBD

7.0 ERROR MESSAGES: The following messages are displayed when errors occur:

JMS_INVALID_CONNECT_ID:	Invalid database connection
JMS_NON_INTERSECT:	AOI does not intersect connected database
JMS_AOI_DESCREPANCY:	AOI structure is invalid

8.0 OTHER APIs: See also JMS_ConfigAOIGet.

9.0 RELATED DOCUMENTATION: Not applicable.

JMS_DataPathnameGet

1.0 FUNCTION: Retrieve the directory path to the requested data type.

2.0 DESCRIPTION: This API returns the paths to the data for a specified data type and a given data base and within the established AOI. The function will return the paths to all the data that intersect the AOI defined for the database connection. The API allocates the necessary memory fro the list to be populated. Therefore the application is responsible for freeing the path list when it is no longer needed.

3.0 SYNTAX:

```
#include  "JMS.h"

#include  "JMS_Errors.h"

tJmsStatus JMS_DataPathnameGet(  tJmsConnection        conn_id ,
                                 tJmsDataType *data_type ,
                                 tJmsPathStruct        *pathlist
```

4.0 ARGUMENTS:

conn_id	Used to identify the unique connection to the SDB.
data_type	Identifies the specific data type such as DCW or CADRG
pathname	Is a character pointer where the directory paths will be placed

5.0 RETURNS: This API returns a status of type tJmsStatus, as defined below, when an error occurs. Otherwise the returned status equals JMS_OPERATION_OK along with the list of directory paths in pathlist.

6.0 DEPENDENCIES: TBD

7.0 ERROR MESSAGES: The following messages are displayed when errors occur:

JMS_INVALID_CONNECT_ID:	Returned when the connection id is not recognized.
JMS_INVALID_DATATYPE :	Returned when the data type is not valid.

8.0 OTHER APIs: See also JMS_ConfigAOISet

9.0 RELATED DOCUMENTATION: Not applicable.

JMS_DbConnect

1.0 FUNCTION: Establish a unique connection to a JMTK geospatial data base.

2.0 DESCRIPTION: JMS_DbConnect is called to create a unique connection to a JMTK geospatial data base that has previously been created through the Spatial Database Manager application. This connection identifies the specific data base that will be accessed and how the SDBM is to interact with that database for all subsequent data requests and/or service calls. Therefore, the JMS_DbConnect function must be called prior to any other Spatial Data Base Module service functions. This includes all but the JMS_DbListGet function in JMTK Version 3.0.

An error is returned if the connection cannot be made. A successful connection will return a unique connection id that is used by other APIs to identify the caller and the database to be accessed. Currently there are no limits to the number of database connection that can be made, although there are practical limits.

When a connection to a specific database can be established, the connection will inherit the databaseÕs AOI as the default AOI. The application can revise the connection AOI with a call to JMS_ConfigAOISet.

When invoked this function allocates memory for the connection to store additional information created by other SDBM API calls. Therefore the JMTK application should call the JMS_DbDisconnect function when the connection is no longer needed so that the reserved memory can be freed.

3.0 SYNTAX:

#include "JMS.h"

#include "JMS_Errors.h"

tJmsStatus JMS_DbConnect(char **name* , tJmsConnection *conn_id*)

4.0 ARGUMENTS:

name A character string identifying a specific geospatial data base. Note that a list of available database names can be retrieved through the API call JMS_DbListGet.

conn_id A variable of the type tJmsConnection that identifies a unique connection to a specific spatial database.

5.0 RETURNS: Returns a status of type tJmsStatus, as defined below, and a unique connection id of type tJmsConnection when the status equals JMS_OPERATION_OK.

6.0 DEPENDENCIES: TBD

7.0 ERROR MESSAGES: The following messages are displayed when errors occur:

JMS_CONNECT_FAILURE_1:	Cannot locate the list of databases
JMS_CONNECT_FAILURE_2:	Cannot read the database list file
JMS_DB_OFFLINE:	Requested database is off-line
JMS_VERSION_MISMATCH:	Software and database versions donÕt match
JMS_OUT_OF_MEMORY:	Out of memory
JMS_NO_DB_SPECIFIED:	A database name was not supplied

8.0 OTHER APIs: See related APIs JMS_DbDisconnect, JMS_ConfigAOISet, and JMS_DbListGet.

9.0 RELATED DOCUMENTATION: Not applicable.

JMS_DbDisconnect

1.0 FUNCTION: Terminate a previous successful database connection.

2.0 DESCRIPTION: MS_DbDisconnect is called to close an existing database connection. This will free any memory that has been allocated to maintain the connection. An error is returned if the connection id is not recognized.

3.0 SYNTAX:

```
#include "JMS.h"

#include "JMS_Errors.h"

tJmsStatus JMS_DbDisconnect( tJmsConnection conn_id )
```

4.0 ARGUMENTS:

conn_id Used to identify the unique connection to the SDB.

5.0 RETURNS: A status of type tJmsStatus is returned. The status will equal JMS_OPERATION_OK when the connection is successfully closed otherwise it will be set to the error status defined below.

6.0 DEPENDENCIES: TBD

7.0 ERROR MESSAGES: The following messages are displayed when errors occur:

JMS_INVALID_CONNECT_ID: returned when the connection id is not recognized.

8.0 OTHER APIs: See also JMS_DbConnect.

9.0 RELATED DOCUMENTATION: Not applicable.

JMS_DbListGet

1.0 FUNCTION: Retrieves a list of spatial database names.

2.0 DESCRIPTION: JMS_DbListGet is called to retrieve a list of available databases that have been created by the Spatial DataBase Manager (SDBM) application. The database names along with their access path are maintained in a private SDBM file. This file is located in the directory defined by environment variable JMS_DBHOME. This function will attempt to access the file, allocate memory to store the list, process the file and return a list of the database names contained within it.

3.0 SYNTAX:

```
#include "JMS.h"

#include "JMS_Errors.h"

tJmsStatus  JMS_DbListGet(   tJmsDbList  **db_list  )
```

4.0 ARGUMENTS:

db_list	A structure pointer that will be populated with the list of available database that have been imported by the SDBM and the number of them. The JMTK application is responsible for freeing the memory allocated for the list when it is no longer needed.

5.0 RETURNS: Returns a status of type tJmsStatus, as defined below, and the list of databases in the argument variable db_list when the status equals JMS_OPERATION_OK.

6.0 DEPENDENCIES: TBD

7.0 ERROR MESSAGES: The following messages are displayed when errors occur:

JMS_INVALID_HOME:	returned when the environment variable JMS_DBHOME cannot be found.
JMS_DBLIST_MISSING:	returned if the SDBM cannot find the database list file.
JMS_DBLIST_ERROR:	returned if the database list file cannot be processed.

8.0 OTHER APIs: None.

9.0 RELATED DOCUMENTATION: Not applicable.

JMS_ErrorGet

1.0 FUNCTION: Retrieve the text message for the specified error code

2.0 DESCRIPTION: This function provides the capability to retrieve the text message for an error code produced by the SDBM. The function accepts the error code and a character pointer to store the message in. The function will allocate memory to store the error message and thus the application programmer is responsible for freeing the memory when it is no longer needed.

3.0 SYNTAX:

```
#include "JMS.h"

#include "JMS_Errors.h"

tJmsStatus JMS_ErrorGet( tJmsStatus error_code, char **error_buff )
```

4.0 ARGUMENTS:

error_code	One of the recognized JMS error statusÕs.
error_buf	A character pointer which will be used to allocate memory and store the returned error message.

5.0 RETURNS: This function will return a status of type tJmsStatus equal to JMS_OPERATION_OK when the error message is successfully returned in error_buf. Otherwise it will return a status as defined below.

6.0 DEPENDENCIES: TBD

7.0 ERROR MESSAGES: The following messages are displayed when errors occur:

JMS_INVALID_ERROR_CODE: returned when the error code is not recognized.

8.0 OTHER APIs: A complete list of the error codes can be found in the public include file JMS_Errors.h

9.0 RELATED DOCUMENTATION: Not applicable.

JMS_InventoryGet

1.0 FUNCTION: Retrieve a data inventory for the specified database connection.

2.0 DESCRIPTION: This API is called to retrieve a list of the data volumes (i.e. DCW, CADRG, WVS, etc.) which are stored within the spatial database identified by the connection id. The calling application will pass the address of the data_list structure to populate. This API will allocate memory for the list. Therefore the JMTK application is responsible for freeing the list when it is no longer needed.

3.0 SYNTAX:

```
#include "JMS.h"

#include "JMS_Errors.h"

tJmsStatus JMS_InventoryGet( tJmsConnection conn_id ,

                             tJmsDbInventory **data_list  )
```

4.0 ARGUMENTS:

conn_id	Used to identify the unique connection to the SDB.
data_list	The address of a structure pointer where the inventory will be placed.

5.0 RETURNS: This API will return a status of type tJmsStatus, as defined below when an error has occurred. If the return status equals JMS_OPERATION_OK the number of data volumes that are within or partially within the established AOI for the database connection will be returned in *data_list.*

6.0 DEPENDENCIES: TBD

7.0 ERROR MESSAGES: The following messages are displayed when errors occur:

JMS_METADATA_PROCESS_ERR:	returned when the metadata cannot be processed.
JMS_INVALID_CONNECT_ID:	returned when the connection is not recognized.

8.0 OTHER APIs: None.

9.0 RELATED DOCUMENTATION: Not applicable.

JMS_MatrixGet

1.0 FUNCTION: Retrieve the matrix data from a spatial database

2.0 DESCRIPTION: This API will retrieve a previously stored matrix file from a spatial database. The application specifies the database containing the matrix through the connection id and the volume it is stored under through the *data_type* parameter. The filename of the matrix data, received from the call to JMS_MatrixPut identifies the specific matrix file to retrieve. The remaining parameters are returned as defined above.

3.0 SYNTAX:

```
#include "JMS.h"

#include "JMS_Errors.h"

tJmsStatus JMS_MatrixGet( tJmsConnection    conn_id,
                          char              *data_type,
                          char              *filename,
                          tJmsArea          *pAoi,
                          int               *row,
                          int               *col,
                          double            *spx,
                          double            *spy,
                          short             *matrix data )
```

4.0 ARGUMENTS:

conn_id	Identifies which database to retrieve the matrix data form.
data_type	Identifies which volume of the specified data base to retrieve the matrix data. This is defined by the application. Any character string is acceptable.
matrix_data	Will contain the actual data retrieved from the database.
pAoi	Will hold the area of interest which encompasses the matrix data.
row	Will contain the number of rows of data within the matrix.
col	Will contain the number of columns of data within the matrix.
spcx	Will contain the spacing in the x direction of the matrix
spcy	Will contain the spacing in the y direction of the matrix

5.0 RETURNS: This API returns a status of type tJmsStatus set to JMS_OPERATION_OK when the requested matrix data is successfully retrieved along with all input arguments populated. When errors are encountered the return status will be set as defined below.

6.0 DEPENDENCIES: TBD

7.0 ERROR MESSAGES: The following messages are displayed when errors occur:

JMS_INVALID_CONNECT_ID: returned when the connection ids is not recognized.

JMS_DATA_READ_ERROR: returned when the matrix cannot be read.

8.0 OTHER APIs: See also JMS_MatrixPut.

9.0 RELATED DOCUMENTATION: Not applicable.

JMS_MatrixPut

1.0 FUNCTION: Save matrix data in a spatial database

2.0 DESCRIPTION: This API will store a JMTK application defined matrix data in a specific spatial database . The application specifies which spatial database to store the matrix in via the connection id. The volume in which the matrix data will be located is defined via the *data_type* parameter. The application must also define the AOI covered by the data, the number of rows and columns along with the spacing in each direction.

The API will validate the parameters received and to store the matrix under the specified volume. If the volume does not exist, it will create it. The filename that the matrix is stored in will be created by the API and returned to the application.

3.0 SYNTAX:

```
#include "JMS.h"

#include "JMS_Errors.h"

tJmsStatus JMS_MatrixPut( tJmsConnection      conn_id,
                          char                *data_type,
                          short               *matrix_data,
                          tJmsArea            *pAoi,
                          int                 row,
                          int                 col,
                          double              spcx,
                          double              spcy,
                          char                *filename)
```

4.0 ARGUMENTS:

conn_id	Identifies which database to store the matrix data in.
data_type	Identifies which volume of the specified data base will receive the matrix data. This is defined by the application. Any character string is acceptable.
matrix_data	Is ten actual data to be stored in the database.
pAoi	Defines the area of interest which encompasses the matrix data.
row	Is the number of rows of data within the matrix.

col	Is the number of columns of data within the matrix.
spcx	Is the spacing in the x direction of the matrix
spcy	Is the spacing in the y direction of the matrix

5.0 RETURNS: The API returns a status of type JmsStatus equal to JMS_OPERATION_OK when the matrix data has been successfully stored in the database along with the filename that data was stored under. Otherwise, an error status is returned.

6.0 DEPENDENCIES: TBD

7.0 ERROR MESSAGES: The following messages are displayed when errors occur:

JMS_INVALID_CONNECT_ID:	returned when the connection id is invalid.
JMS_IMPORT_DATA_ERROR:	returned when the matrix cannot be stored.
JMS_MALLOC_ERROR:	returned when memory cannot be allocated.

8.0 OTHER APIs: See also JMS_MatrixGet.

9.0 RELATED DOCUMENTATION: Not applicable.

JMS_MetadataGet

1.0 FUNCTION: Retrieve metadata for a database, a data volume or a dataset within a spatial database.

2.0 DESCRIPTION: JMS_MetadataGet is called to retrieve metadata that is maintained for the database, the data volumes or the datasets. The first two arguments are mandatory. The JMTK application must identify the database to retrieve metadata from by providing a connection id. The application must also identify which level of metadata is being requested. Level 1 is associated with the database metadata providing information relative to the entire database. Level 2 metadata contains information for a specific data volume (i.e. DCW, WVS. CADRG, DTED, etc.). Level 3 describes the actual datasets in a data volume.

The next two arguments, *data_type* and *data_set* are optional depending upon the level of metadata requested. If the application is requesting Level 1 (database metadata) then both *data_type* and *data_set* are ignored and the application can pass a NULL in these locations. If the application request Level 2 metadata, then *data_type* is mandatory and *data_set* is ignored. A Level 3 request requires both the *data_type* and *data_set* to be populated.

The next argument is a variable argument list that is terminated by a NULL. This variable list contains pairs of Jms Metadata resources followed by the variable or structure to populate. The application must pass a pointer to the variable or structure. The API will allocate space, as appropriate so the application is responsible for freeing the memory when it is no longer needed.

3.0 SYNTAX:

#include "JMS.h"

#include "JMS_Errors.h"

tJmsStatus JMS_MetadataGet(tJmsConnection *conn_id*, int level,

char **data_type*, char **data_set*,

va_list ,

NULL)

4.0 ARGUMENTS:

conn_id: Identifies which database to access.

level Identifies the type of metadata to retrieve where LVL1=database, LVL2=volume, LVL3=dataset.

data_type Is the name of the data volume to access if the level is LVL2 otherwise this argument should be NULL.

data_set Is the name of the dataset within the data volume that is to be accessed if the level is set to LVL3, otherwise this argument should be set to NULL.

va_list Is a variable argument list containing pairs of resources and appropriate variables to store them in. This list is terminate by a NULL.

5.0 RETURNS: This API return will return an error status of the type tJmsStatus, as defined below, when and error is encountered. Otherwise, the passed variables pointers are populated and the return status will equal JMS_OPERATION_OK.

6.0 DEPENDENCIES: TBD

7.0 ERROR MESSAGES: The following messages are displayed when errors occur:

JMS_CONNECTIONID_ERROR:	Connection id error.
JMS_LEVEL_ERROR:	Level error.
JMS_UNKNOWN_METADATA_TYPE:	Unknown metadata type.
JMS_UNKNOWN_DATA_TYPE:	Unknown data type.
JMS_UNKNOWN_DATASET:	Unknown dataset.
JMS_INVALID_RESOURCE_LIST:	Invalid resource for level
JMS_METADATA_PROCESS_ERR:	Metadata process error.

8.0 OTHER APIs: A list of all the Jms resources is provided in the public include file JMS.h

9.0 RELATED DOCUMENTATION: Not applicable.

JMV_LoadMap

JMV_UnLoadMap

FUNCTION

Load and unload map data

SYNOPSIS

```
#incldue <JMV/Load.h>

JMV_MapObjectID
JMV_LoadMap(
        p_str    pMapDataPath,
    p_str    pOutputDir,
    JMV_Atom lAtom
    );

int
JMV_UnLoadMap(
        p_str          pOutputDir,
    JMV_MapObjectID MapId
    );
```

DESCRIPTION

JMV_LoadMap() registers the map data products found in the argument pMapDataPath with the JMTK Visual domain server. All associated map data product indices are created under the directory specified in the argument pOutputDir. The map data type is specified by the argument lAtom. The following map data product type convenience macros are provided:

```
JMV_ATOM_DTED
JMV_ATOM_RPF
JMV_ATOM_VPF
```

JMV_UnLoadMap() unregisters the map data product specified by the argument lMapId and removes any indices in the argument pOutputDir.

RETURN VALUE

JMV_LoadMap() returns a JMV_MapObjectID. The define constant BAD_OBJECT is returned upon failure.
JMV_UnLoadMap() returns a boolean value, UNLOAD_SUCCESS or UNLOAD_FAILURE

MAbortAnimation

FUNCTION

Abort the animation of an object.

SYNTAX

C Interface

```
void MAbortAnimation( channel, window )
        Channel   channel;
        WindowId  window;
```

ARGUMENTS

channel	The connection to Cartographer; returned from MOpenChannel().
window	The window where animation is currently under way.

DESCRIPTION

MAbortAnimation aborts the animation of an object. If the current animation is a create type, then the object is removed from the screen, and the object is freed. If the animation is a modify type, then the object is left as it was before the animation started.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

An invalid window id was used.

SEE ALSO

MCreateObject(3C), MCreateText(3C), MModifyObject(3C)

MAbortMap

FUNCTION

Abort current map draw command.

SYNTAX

C Interface

```
void MAbortMap( channel, window )
        Channel   channel;
        WindowId  window;
```

ARGUMENTS

channel	The connection to the Chart Manager; returned from MOpenChannel.
window	The window to which the abort is directed.

DESCRIPTION

MAbortMap aborts the current draw request, if one is currently being processed. The abort map is filtered to all connected Draw Modules which are processing a request for this map. If the map has already been partially drawn (eg. the new area is being viewed), then only those maps which have not completed are aborted. Depending on the state of the map system when the abort is requested, one of two things can occur:

(1) If a draw is already partially complete, then those maps which have been already drawn are preserved, and a ChangeMapNotify or UpdateMapNotify is sent to all interested Clients.

(2) If no maps have yet been drawn, then the request is completely abandoned, and the previous map is preserved. In this case, an AbortMapNotify is sent to all interested Clients.

In both cases, the library error handler for the Client requesting the change map request, or in the case of an internal request, the Client which owns the window, will receive an error notification with value MapDrawAborted.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

An invalid window id was used.

MapDrawAborted

Sent automatically to the Client originally making the change request, or else sent to the window's owner.

SEE ALSO

MChangeMap(3Map), MEvents(3C)

MAddFeature

FUNCTION

Add specified feature to a map.

SYNTAX

C Interface

```
void MAddFeature( channel, window, feature )
Channel   channel;
WindowId  window;
        MapFeatureAttributes     *feature;
```

ARGUMENTS

channel	The connection to the Chart Manager; returned from MOpenChannel.
window	The window on which the feature is to be displayed.
feature	A map feature to be added to the map. See MFeatAtts(3Map) for information on this structure's contents.

DESCRIPTION

MAddFeature adds the specified feature to the currently drawn map. Some map products may not support feature overlays. A FeatureNotSupported error occurs in this case. Also some overlay features may not be available on all servers. In this case, a FeatureNotAvailable error occurs. The MListFeatures(3Map) call provides a list of supported features in the server.

If the specified feature or features are already present on the display list then this call will modify the feature's attributes according to the attributes specified in the FeatureAttributes record. If the feature is not present on the display list, then it is added to it. Note that the sub type field supports the wildcard value AnyFeature to specify an entire class of features to be added.

ERRORS

See MAddFeatures for a synopsis of possible errors.

SEE ALSO

MAddFeatures(3Map), MChangeMap(3Map), MFeatAtts(3Map), MFeatMask(3Map), MModifyFeature(3Map), MModifyFeatures(3Map), MRemoveFeature(3Map), MRemoveFeatures(3Map), MuReference(3Mu)

MAddFeatures

FUNCTION

Add specified features to a map.

SYNTAX

C Interface

```
void MAddFeatures( channel, window, feature, nfeatures )
        Channel   channel;
        WindowId  window;
                MapFeatureAttributes     *features;
        int  nfeatures;
```

ARGUMENTS

channel — The connection to the Chart Manager; returned from MOpenChannel.

window — The window on which the features are to be displayed.

features — A list of map features to be added to the map. See MFeatAtts(3Map) for information on the contents of each record.

nfeatures — The size of the feature list specified in feature.

DESCRIPTION

MAddFeatures adds the specified features to the currently drawn map. Some map products may not support feature overlays. A FeatureNotSupported error occurs in this case. Also some overlay features may not be available on all servers. In this case, a FeatureNotAvailable error occurs. The MListFeatures(3Map) call provides a list of supported features in the server. Note that this call should not be used to modify existing features, but simply to add new features.

If the specified feature or features are already present on the display list then this call will modify the feature's attributes according to the attributes specified in the FeatureAttributes record. If the feature is not present on the display list, then it is added to it. Note that the sub type field supports the wildcard value AnyFeature to specify an entire class of features to be added.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

An invalid window id was used.

BadValueError

An invalid or non-existent feature value was specified.

ErrorDrawingFeature

An error occurred while drawing the specified feature, which is described in the accompanying MapErrorCodeIn- formation structure.

FeatureNotSupported

The specified feature is not supported on the current map. An accompanying MapErrorCodeInformation structure specifies the unsupported feature.

FeatureNotAvailable

The specified map feature is not available. This is probably because no feature generator exists on-line for drawing it. The accompanying MapErrorCodeInforma tion structure specifies the unavailable feature.

MaxScale

The current view is zoomed too far out to display the given feature. The feature product generating the error is provided in an accompanying MapErrorCodeInfor- mation structure.

MinScale

The current view is zoomed too far in to display the given feature. The feature product generating the error is provided in an accompanying MapErrorCodeInfor- mation structure.

SEE ALSO

MAddFeature(3Map), MChangeMap(3Map), MFeatAtts(3Map), MFeatMask(3Map), MModifyFeature(3Map), MModifyFeatures(3Map), MRemoveFeature(3Map), MRemoveFeatures(3Map), MuReference(3Mu)

MAddInput

FUNCTION

Add an input source to the context.

SYNTAX

```
void MAddInput( fd, mask, proc, data )
        int  fd;
        int  mask;
        MCallbackProc  proc;
        Pointer   data;
```

ARGUMENTS

fd	Specifies the source file descriptor on a Unix based system.
mask	Specifies the condition mask that tells when the routine should be called. Valid entries for this field are ServiceReadMask, ServiceWriteMask, Ser- viceExceptMask, or ServiceAllMask.
proc	Specifies the procedure that is to be called when the condition is satisfied
data	Specifies the argument that is to be passed to the specified procedure when the callback is called.

DESCRIPTION

The MAddInput routine registers with the Service manager a source of data that is to be monitored. The data source can be any sink or source of data that has an associated file descriptor. The source is monitored for a given condition, and when the condition is met the specified routine is called.

STRUCTURES

C Interface

```
typedef void *MCallbackProc();
```

A CallbackProc is the name of a function to specify as a callback procedure.

ERRORS

BadValueError

The file descriptor for a channel has been specified. You cannot specify an input handler for a Chart communications channel, as this would overwrite the input

handler for Chart.

SEE ALSO

MRemoveInput(3Map), MSetEventHandler(3Map)

MAddObject

FUNCTION

Add an object to a list or class.

SYNTAX

C Interface

```
void MAddObject( channel, list, object )
Channel   channel;
ObjectId  list;
ObjectId  object;
```

ARGUMENTS

channel — The connection to Cartographer; returned from MOpenChannel.

list — The list or class to which the object will be added.

object — The object to be added to the list.

DESCRIPTION

MAddObject adds the specified object to the list or class. The attributes of the object remain unchanged, it simply becomes a member of the list.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid object id was used.

SEE ALSO

MRemoveObject(3C), MDestroyList(3C), MDestroyObject(3C), MExchangeObject(3C)

MAddPoint

FUNCTION

Add a single point to the end of a polyline or polygon.

SYNTAX

C Interface

```
void MAddPoint( channel, object, points, location, max points )
        Channel         channel;
        ObjectId  object;
        MapPoint  *points;
        int  location;
        int  max points;
```

ARGUMENTS

channel — The connection to Cartographer; returned from MOpenChannel.

object — The location for which to place the segment.

points — The point or points to be added to the Polygon or Polyline.

location — The place to insert the points. This value represents the index of the vertex where the points are to be inserted. A negative value indicates to insert the points at the beginning of the polygon or polyline, and a value greater than or equal to the current number of points indicates to append the points.

max points — The number of points being inserted.

DESCRIPTION

MAddPoint inserts one or more points into a Polyline or Polygon object, without the overhead of destroying and creating the object. This command can be used to append points to the end of a polygon or polyline by providing a location value which exceeds the current number of points in the object.

Using this command on objects that are not of type Polyline or Polygon will cause a BadValueError.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId
An invalid object id was used.

BadValueError
The object is not a Polyline or Polygon object.

SEE ALSO
MDrawPolyLine(3C), MDrawPolygon(3C), MSetObjectData(3C)

MAddProduct

FUNCTION

Add specified map product to a map.

SYNTAX

C Interface

```
void MAddProduct( channel, window, product )
Channel   channel;
WindowId  window;
        MapProductAttirbutes     *product;
```

ARGUMENTS

channel	The connection to the Chart Manager; returned from MOpenChannel.
window	The window on which the product is to be displayed.
product	A map product to be added to the map display.

DESCRIPTION

MAddProduct adds the specified product to the currently drawn map. Some map products may not be drawable in the current display. This routine uses the current boundary attributes, and color criteria to display the given map product. The product is overlayed onto the existing geographic display, and may occlude products which have been previously drawn. If the map product has already been drawn, then the request is simply ignored. A description of the product attributes is found on the MProductAttributes man page.

ERRORS

See MAddProducts for a synopsis of possible errors.

SEE ALSO

MAddProducts(3Map), MChangeMap(3Map), MRemoveProduct(3Map), MRemoveProducts(3Map), MuReference(3Mu)

MAddProducts

FUNCTION

Add the specified map products to a map.

SYNTAX

C Interface

```
void MAddProducts( channel, window, products, nproducts )
    Channel   channel;
    WindowId  window;
        MapProductAttributes     *products;
    int  nproducts;
```

ARGUMENTS

channel — The connection to the Chart Manager; returned from MOpenChannel.

window — The window on which the products are to be displayed.

products — A list of map products to be added to the map display.

nproducts — The number of products to be added.

DESCRIPTION

MAddProducts adds the specified list of map products to the currently drawn map. Some map products may not be drawable in the current display. This routine uses the current boundary attributes, and color criteria to display the supplied map products. The products are overlayed onto the existing map display, and may occlude products which have been previously drawn. If the map products have already been drawn, then the request is simply ignored. A description of the product attributes is found on the MProductAttributes man page.

ERRORS

AlreadyDrawingMap
: A map draw command is already in progress for the specified window.

BadChannel
: An invalid channel id was used.

BadMapEntry
: An error occurred while drawing specified map product,

due to bad data format, missing files, or other related problems. The MapErrorCodeInformation structure indicates the product generating the error.

BadWindowId
An invalid window id was used.

BadValueError
An invalid or non-existent map product was specified.

DuplicateMap
The specified map product is already being partially or fully displayed. The duplicate portion is ignored; however, additional draw requests occur for the part which hasn't been drawn.

ErrorDrawingMap
An error occurred for a specific draw request.

MapDrawAborted
The specified draw request was aborted at the request of some Chart Client.

MapTooSmall
The specified boundary results in an image which is too small to draw.

ProductNotFound
The specified request resulted in no map products being drawable which match the given product(s).

ProjectionNotSupported
The specified product is not supported in the currently drawn projection.

SystemNotSupported
The specified product is not supported in the current earth model.

TooManyMaps
The specified product results in more than MAX_MAPS draw requests if fully satisified. At most MAX_MAPS maps may supported inside a map window at once. The net effect is to draw up to MAX_MAPS maps, and trim requests beyond this.

SEE ALSO
MAddProduct(3Map), MChangeMap(3Map), MRemoveProduct(3Map), MRemoveProducts(3Map), MuReference(3Mu)

MAddTimeOut

FUNCTION

Register a timeout with the service mananger.

SYNTAX

```
#include <M/Service.h>

MapTimerId MAddTimeOut( proc, data, delay ) MTimerCallbackProc  proc;
        Pointer   data;
        int  delay;
```

ARGUMENTS

proc — Specifies the procedure that is to be called when time expires.

data — Specifies the argument that is to be passed to the specified procedure when the callback is called.

delay — Specifies the time interval in milliseconds.

DESCRIPTION

The MAddTimeOut() routine registers with the Service manager an interval timer and returns an identifier for it. The callback procedure, proc is called when the timer elapses, and then the timeout is removed. The returned id can be used to remove the time out before the timer has elapsed, with the call, MRemoveTimeOut.

RETURN

This function returns a MapTimerId. The value will be -1 if the call fails.

STRUCTURES

C Interface

```
typedef int MapTimerId;
typedef void *MTimerCallbackProc();
```

The MapTimerId is an integer identification of this timeout request. It can be used to remove a timeout using the MRemoveTimeout call. The MTimerCallbackProc describes the name of a procedure to call periodically.

SEE ALSO

MRemoveTimeOut(3Map)

MAddVolume

FUNCTION

Add a new volume to the Map Search Path.

SYNTAX

C Interface

```
void MAddVolume( channel, volume )
        Channel  channel;
        char     *volume;
```

ARGUMENTS

channel	Specifies the connection to Chart returned by MOpenChannel.
volume	The volume to be added to the map search path.

DESCRIPTION

The MAddVolume function adds a new volume to the Map Search Path. All map data files contained in this volume will be available for display in the server. If the environment variable MapNoRecursion is set, then only the data files contained in the directory are examined, otherwise all of the subdirectories (if any) are also examined.

When a volume is added, the files in that directory are checked to see if they contain valid map headers. If a file does, then a Draw Module is identified which is then responsible for drawing the map. If no Draw Module is found which is capable of drawing this map, then the description is stored, pending any future Draw Modules which attach to Chart.

ENVIRONMENT

MapNoRecursion

When this environment variable is set, only the specified path is checked for map files. Otherwise the specified path and all of its subdirectories are checked.

ERRORS

BadChannel

An invalid channel id was used.

BadVolume

The specified volume is invalid.

SEE ALSO

MReloadSearchPath(3Map), MRemoveVolume(3Map), MApplyAttributes(3C)

MApplyAttributes

FUNCTION

Set the attributes of an object and its children.

SYNTAX

C Interface

```
void MApplyAttributes( channel, object, atts,
    value mask )
        Channel   channel;
        ObjectId  object;
        ObjectAttributes    *atts;
        MapValueMask   value mask;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

object — The object whose attributes will be changed.

atts — The object attributes to be applied to the object.

value mask — A value mask representing the attributes to be modified.

DESCRIPTION

MApplyAttributes sets the attributes of an object and its children (if any) to the attributes specified. If the Object is a Drawable, its attributes will be updated and the changes will be visible on the screen (assuming the drawable is currently visible). Since a drawable cannot have any children, this call would have the same effect as MSetAttributes(). If the Object is a Class, the attributes of the class as well as the members of the Class are set to the specified attributes. If the object is a List, all of the objects in the List will be updated. Finally, if the object is a Template, the attributes of the template will be updated, and no changes will be seen on the screen. This would have the same effect as MSetAttributes(), since a Template cannot have children.

The values for the ObjectAttributes fields are taken from the parameter atts if the corresponding bit in the value mask is set. Otherwise, the value is left unchanged.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId
An invalid object id was used.

BadValueError
An invalid field value was specified in the ObjectAt- tributes structure.

SEE ALSO
MApplyColor(3C), MApplyData(3C), MApplyFillType(3C), MApplyFont(3C), MApplyPickability(3C), MApplyVisibility(3C), MObjAtts(3C), MObjMask(3C), MSetAttributes(3C)

MApplyColor

FUNCTION

Change the color of an object and its children.

SYNTAX

C Interface

```
void MApplyColor( channel, object, color )
Channel   channel;
ObjectId  object;
char *color;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

object — The id of the object whose color is to be set.

color — The new color for the object.

DESCRIPTION

MApplyColor sets the color of an object and all of its children to a named color from the RGB Database. Behaviorally, this call works in a manner similar to MApplyAttributes(). The color must exist in the X Windows RGB Database. If it does not, the color is not changed, and an error message is generated.

ERRORS

BadChannel
: An invalid channel id was used.

BadObjectId
: An invalid object id was used.

BadValueError
: An invalid color was specified.

ColorTableFull
: The specified color cannot be realized because the color table is full.

SEE ALSO

MApplyAttributes(3C), MSetColor(3C)

MApplyData

FUNCTION

Set the client_data field of an object and its children.

SYNTAX

C Interface

```
void MApplyData( channel, object, client data )
Channel   channel;
ObjectId  object;
char *client data;
```

ARGUMENTS

channel — The connection to Cartographer; returned from MOpenChannel.

object — The id of the object whose data attribute is to be set.

client data — The value to store in the client_data field.

DESCRIPTION

MApplyData sets the data field of an object and all of its children to the specified value. Behaviorally this call works in a similar manner to MApplyAttributes(). The client_data field can be any 32 bit data item. This item is returned back to the user when it is requested through MQueryObject, or by an ObjectSelectEvent.

ERRORS

BadChannel
: An invalid channel id was used.

BadObjectId
: An invalid object id was used.

SEE ALSO

MApplyAttributes(3C) MSetData(3C)

MApplyFillOffset

FUNCTION

Set the pixel offset of an object and its children.

SYNTAX

C Interface

```
void MApplyFillOffset( channel, object, fill offset ) Channel   channel;
        ObjectId  object;
        int  fill offset;
```

ARGUMENTS

channel — The connection to Cartographer; returned from MOpenChannel.

object — The id of the object whose fill offset is to be set.

fill offset — The new pixel fill offset for the object.

DESCRIPTION

MApplyFillOffset sets the fill offset of an object and all of its children. Behaviorally, this call works in a manner similar to MApplyAttributes(). This value provides a starting pixel offset for objects using FillTransparent fill type. Refer to MObjAtts(3C) for more information on this field.

ERRORS

BadChannel
: An invalid channel id was used.

BadObjectId
: An invalid object id was used.

BadValueError
: An invalid fill offset was specified.

SEE ALSO

MApplyAttributes(3C), MObjAtts(3C), MSetFillOffset(3C)

MApplyFillType

FUNCTION

Set the fill type of an object and its children.

SYNTAX

C Interface

```
void MApplyFillType( channel, object, fill type ) Channel   channel;
                     ObjectId  object;
                     MapFillType    fill type;
```

ARGUMENTS

channel The connection to Cartographer; returned from MOpenChannel.

object The id of the object whose fill type is to be set.

fill type The new fill type of the object.

DESCRIPTION

MApplyFillType sets the fill type of an object and all of its children. Behaviorally, this call works in a manner similar to MApplyAttributes(). Valid values for fill type are: FillEmpty, FillOpaque, FillTransparent, FillDotted, FillHorizontalStripes, FillVerticalStripes, FillNega- tiveSlants, FillPositiveSlants, and FillCrossHatch. If the fill type is FillOpaque, then the object is filled completely with the color of the object. If the fill type is FillEmpty, then only the border of the object is drawn. If the fill type is FillTransparent, then the object is filled with a pattern that allows the user to see the map through the object. The pattern which is drawn is a function of the fill weight and fill offset fields. Refer to MObjAtts(3C) for more information on this field.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid object id was used.

BadValueError

An invalid fill type was specified.

SEE ALSO

MApplyAttributes(3C), MObjAtts(3C), MSetFillType(3C)

MApplyFillWeight

FUNCTION

Set the fill weight of an object and its children.

SYNTAX

C Interface

```
void MApplyFillWeight( channel, object, fill weight ) Channel   channel;
                       ObjectId  object;
                       int  fill weight;
```

ARGUMENTS

channel — The connection to Cartographer; returned from MOpenChannel.

object — The id of the object whose fill weight is to be set.

fill weight — The new fill weight of the object.

DESCRIPTION

MApplyFillWeight sets the fill weight of an object and all of its children. Behaviorally, this call works in a manner similar to MApplyAttributes(). The fill weight attribute has no effect on the object unless the fill type is set to FillTransparent. Refer to MObjAtts(3C) for more information on how this works.

ERRORS

BadChannel
: An invalid channel id was used.

BadObjectId
: An invalid object id was used.

BadValueError
: An invalid fill weight was specified.

SEE ALSO

MApplyAttributes(3C), MObjAtts(3C), MSetFillWeight(3C)

MApplyFont

FUNCTION

Set the font of an object and its children.

SYNTAX

C Interface

```
void MApplyFont( channel, object, font )
Channel   channel;
ObjectId  object;
char *font;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

object — The id of the object whose font is to be set.

font — The new name of the font for the object.

DESCRIPTION

MApplyFont sets the font of the object and all of its children to the named font. Behaviorally, this call works in a similar manner to MApplyAttributes(). If the font name is not valid, the font remains unchanged, and an error message is issued. If the object is not of type Text, Symbol, or Char, there is no real change as these are the only three object types that use the font parameter. All font names must be valid X Window font names.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid object id was used.

BadValueError

An invalid font name was specified.

SEE ALSO

MApplyAttributes(3C), MSetFont(3C)

MApplyHiLite

FUNCTION

Highlight/Unhighlight an object in a window.

SYNTAX

C Interface

```
void MApplyHiLite ( channel, object, hilite)
Channel   channel;
ObjectId  object;
Boolean   hilite;
```

ARGUMENTS

channel Specifies the connection to Cartographer; returned from MOpenChannel.

object Specifies the object to be highlighted.

hilite Specifies the highlight state. When set to True, then the object will be highlighted; when set to False, the object will be unhighlighted.

DESCRIPTION

The MApplyHiLite causes the object and all of its children to be displayed in the current highlight color. If object is of type Drawable or Template, then this call is the same as MSetHiLite. If object is of type List, then this call will change the highlight state for the list and all of its children. If object is of type Class, then the highlight states of the Class and all of its members are modified. The color of highlighted objects can be changed with the routine MSetHiLiteColor.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid object id was used.

SEE ALSO

MApplyAttributes(3C), MObjAtts(3C), MSetHiLite(3C), MSetHiLiteColor(3C)

MApplyLineStyle

FUNCTION

Set the line style of an object and its children.

SYNTAX

C Interface

void MApplyLineStyle(channel, object, line style) Channel channel;
ObjectId object;
MapLineStyle line style;

ARGUMENTS

channel Specifies the connection to Cartographer; returned from MOpenChannel.

object The object whose line style attribute is to be changed.

line style
The new line style for the object.

DESCRIPTION

MapApplyLineStyle sets the style of the line. The line style determines whether the line is solid, dashed, or double dashed. Behaviorally, this call works in a manner similar to MApplyAttributes(). Valid line styles are MapLineSo- lid, MapLineDashed, and MapLineDoubleDashed.

ERRORS

BadChannel
An invalid channel id was used.

BadObjectId
An invalid object id was used.

BadValueError
An invalid line style was specified.

SEE ALSO

MApplyAttributes(3C), MSetLineStyle(3C)

MApplyLineType

FUNCTION

Set the line type of an object and its children.

SYNTAX

C Interface

void MApplyLineType(channel, object, line type) Channel channel;
ObjectId object;
MapLineType line type;

ARGUMENTS

channel Specifies the connection to Cartographer; returned from MOpenChannel.

object The object whose line type attribute is to be changed.

line type The new line type for the object.

DESCRIPTION

MApplyLineType sets the line type of an object and all of its children. Behaviorally, this command works in a similar manner to MApplyAttributes(). The line type determines how a line will be drawn on the earth. Geodesic, GreatCircle, and RhumbLine are valid line types. If the line type is Geodesic, the default value, a line is drawn between the two points after they are projected, the exact path that the line crosses will vary depending on the projection. If the line type is GreatCircle, a line is drawn between the two points following a great circle; this will be the shortest path between the two points. In the last case, RhumbLine, the two points are connected with a line of constant bearing. In the latter two cases (GreatCircle and RhumbLine), the lines may not be straight, depending on the projection.

ERRORS

BadChannel
An invalid channel id was used.

BadObjectId
An invalid object id was used.

BadValueError
An invalid line type attribute was used.

SEE ALSO

MApplyAttributes(3C), MSetLineType(3C),

MApplyLineWidth

FUNCTION

Set the line_width of an object and its children.

SYNTAX

C Interface

```
void MApplyLineWidth( channel, object, line width ) Channel   channel;
                      ObjectId  object;
                      int  line width;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

object — The object whose line width attribute is to be changed.

line width — The new line width for the object.

DESCRIPTION

MApplyLineWidth sets the line width of an object and all of its children. Behaviorally, this command functions in a manner similar to MApplyAttributes().

ERRORS

BadChannel
: An invalid channel id was used.

BadObjectId
: An invalid object id was used.

BadValueError
: An invalid line width was specified.

SEE ALSO

MApplyAttributes(3C), MSetLineWidth(3C)

MApplyPickability

FUNCTION

Set the pickability of an object and its children.

SYNTAX

C Interface

```
void MApplyPickability( channel, object, pickability ) Channel   channel;
                                                        ObjectId  object;
                                                        Boolean   pickability;
```

ARGUMENTS

channel — The connection to Cartographer; returned from MOpenChannel.

object — The id of the object whose pickability attribute is to be set.

pickability — The new pickability of the object.

DESCRIPTION

MApplyPickability sets the pickability of an object, and all of its children. If object is a List, then the pickability of the List and all of its child objects are set. If object is a Class, then all of the class members have their pickability modified. If object is a Template or a Drawable, then this call is identical to MSetPickability. Valid values for the pickability parameter are Pickable (True) and NotPickable (False).

The pickability determines whether or not an object is selectable on the window. In the simplest case (an object that is not a member of a list), if the object is selected and its pickability is set to either Pickable or ParentPick- able, then an ObjectSelectEvent is sent to the Client that owns the object. If the object is NotPickable, then no event is sent.

If an object is a member of a list, and is Pickable an ObjectSelectEvent is sent to the Client, just as if it were not in a list. If the object is not pickable, then no event is sent. If the object's pickability is set to ParentPick- able then the pickability of the object's parent is checked. If this is pickable, then an ObjectSelectEvent is sent with the id of the list (not the id of the object actually

selected). If the list is NotPickable, then no event is sent, and if the list is ParentPickable, then this process continues until an object is found that is not pickable, or the top of the object tree is reached. If the top list in the object tree is ParentPickable, then an event is sent to the Client just as if this object were set to Pickable.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid object id was used.

BadValue

An invalid value for pickability was specified.

SEE ALSO

MApplyAttributes(3C), MSetPickability(3C)

MApplyPixel

FUNCTION

Change the color of an object using the pixel value.

SYNTAX

C Interface

```
void MApplyPixel( channel, object, pixel )
Channel   channel;
ObjectId  object;
unsigned long  pixel;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

object — The id of the object whose color is to be changed.

pixel — The pixel value of the color for the object.

DESCRIPTION

MApplyPixel sets the color of a single object to the specified pixel value. Behaviorally, this call works in a similar manner to MApplyAttributes(). The pixel value must be a valid pixel value for the display, and should be a read-only allocated color.

ERRORS

BadChannel
: An invalid channel id was used.

BadObjectId
: An invalid object id was used.

BadValueError
: An invalid pixel value was specified.

SEE ALSO

MApplyColor(3C), MObjAtts(3C), MSetAttributes(3C),

MApplyTemplate

FUNCTION

Copy a template into an object and all of its children.

SYNTAX

C Interface

```
void MApplyTemplate( channel, object, template,
value mask );
    Channel   channel;
    ObjectId  object;
    ObjectId  template;
    MapValueMask   value mask;
```

ARGUMENTS

channel Specifies the connection to Cartographer; returned from MOpenChannel.

object The object whose attributes are to be changed.

template The template to be applied.

value mask
A value mask representing the attributes to be applied.

DESCRIPTION

MApplyTemplate applies the object attributes specified by template to the object or group of objects represented by object. Only the attributes whose bit is set in the value_mask will be set in the objects. The effect of this call is nearly identical to MApplyAttributes, with the exception that MApplyAttributes takes its attributes from an ObjectAttributes structure, whereas MApplyTemplate copies the attributes from an existing template.

ERRORS

BadChannel
An invalid channel id was used.

BadObjectId
An invalid object id or template ID was used.

SEE ALSO

MApplyAttributes(3C), MSetTemplate(3C) MObjAtts(3C)

MApplyVisibility

FUNCTION

Set the visibility of an object and its children.

SYNTAX

C Interface

```
void MApplyVisibility( channel, object, visibility ) Channel   channel;
                       ObjectId  object;
                       Boolean   visibility;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

object — The object whose visibility is to be changed.

visibility — The new visibility state for the object.

DESCRIPTION

MapApplyVisibility sets the visibility of an object and all of its children. Valid values for visibility are Visible (True) and Hidden (False). If an object's visibility is set to Hidden, then the object is not visible on the map. If the object is not a List or a Class, this routine is the same as MSetVisibility.

ERRORS

BadChannel
: An invalid channel id was used.

BadObjectId
: An invalid object id was used.

BadValueError
: An invalid visibility state was specified.

SEE ALSO

MApplyAttributes(3C), MSetVisibility(3C), MObjAtts(3C)

MChangeMap

FUNCTION

Change the maps and features displayed in a map window.

SYNTAX

C Interface

```
void MChangeMap( channel, window, atts, value mask ) Channel   channel;
                 WindowId  window;
                 MapChangeAttributes *atts;
                 MapValueMask   value mask;
```

ARGUMENTS

channel — The connection to the Chart Manager; returned from MOpenChannel.

window — The window whose map is to be changed.

atts — The structure from which the values (as specified by the value mask) are to be taken. The value mask should have the appropriate bits set to indicate which attributes have been set in the structure.

value mask — Which attributes of the map are to be changed. The mask is the bitwise inclusive OR of the valid attribute mask bits. If value mask is zero (CMSetNone), the attributes are ignored, and the value of that field for the currently displayed map is used instead. Valid values for value mask include:

CMSetBoundary: The map boundaries are changed according to the value of MapBoundaryAttributes.

CMSetProjection: The map's projection is changed to the value specified in the ProjectionType.

CMSetProduct: A new map display is drawn according to the list of MapProductAttributes.

CMAddProduct: The current map display is updated with the given list of MapProductAttributes. The requested set is overlayed onto the existing map if they do not already exist.

Note 1: if neither CMSetProduct nor CMAddProduct is set, then the draw request derives the list of maps to be drawn based upon the existing set of map products.

Note 2: CMAddProduct is mutually exclusive with CMSetProduct.

CMSetColors: The map's color and/or intensity is changed according to the list of provided MapColorAttributes.

CMSetFeature: The map's feature overlays are drawn according to the list of provided Map- FeatureAttributes.

CMAddFeature: The indicated set of features is added to the current set of displayed features.

Note 1: if neither CMSetFeature nor CMAddFeature is set, then the draw request derives the list of features to be drawn based upon the last specified list of Map- FeatureAttributes, from a MChangeMap() which had the CMSetFeature or CMAddFeature set, or else by way of an MAddFeature(3Map) or MAddFeatures(3Map) call.

Note 2: CMAddFeature is mutually exclusive with CMSetFeature.

CMSetAll: Changes all attributes of the map to the new values specified in the MapChangeAttri- butes structure.

DESCRIPTION

The MChangeMap function changes the map display in the specified window. This function forces the window to be changed to the attributes that are set in MapChangeAttri butes.

The Chart Manager receives this request and formulates one or more draw requests to be processed by Draw Modules. Periodic updates, in the form of events, are provided for Clients interested in the progress of a change map request.

Error processing for a change map request can take on two forms: synchronous and asynchronous. Synchronous errors are fed back immediately to the Client making the change map request. An example of such an error is BadValueError, which occurs when one of the specified fields is illegal. Asynchronous errors occur at some time after the Client returns from the MChangeMap call. These errors occur as the result of some problem encountered by the Chart Manager, or one of the Draw Modules, while the request is being serviced. These errors are also fed back to the Client making the request via its error handler.

Errors which occur while a change map request is in progress can be either fatal or non-

fatal. Fatal errors result in the map request being canceled, and an AbortMapNotify being sent to all interested Clients. The most common case where this occurs is when all of the map products being requested are denied, resulting in no map products being drawn. Nonfatal errors which occur include the inability to draw one particular map product. These errors are fed back to the error handler of the Client making the request. Some of these errors have associated data. See the ERRORS section below, as well as MError(3C) for more detail.

In the case where the request being made is internally generated, say, for example, a zoom box request, the errors are then fed back to the process which owns the window (eg, the Chart Client which created the window).

STRUCTURES

C Interface

```
typedef struct {
        int   num_products;
        MapProductAttributes products[MAX_MAPS]; ProjectionType
        projection; MapBoundaryAttributes     boundary;
                        int                num_colors;
        MapColorAttributes       color[MAX_COLOR_MODELS]; int
        num_features; MapFeatureAttributes *features;
} MapChangeAttributes;
```

The fields for the MapChangeAttributes structure are described below:

products

A list of map product specifications. This structure is described in detail under MProdAtts(3Map). These values are used only if the CMSetProduct bit is set in value mask.

num_products

The number of products specified in the products list. This value cannot exceed the Chart Manager constant MAX MAPS.

projection

A projection to be displayed. The MProjection(3Map) man page discusses display projections in detail. This value is used only if the CMSetProjection bit is set in value mask.

Note: not all map products can be displayed in all projections; hence some projections may not be able to simulataneously display two different map products. The value AnyProjection allows the Chart Manager to determine the projection most common to all of the map products.

color

A list of color specifications for the map display. The MColor(3Map) man page discusses color models in detail. This specification is used only if the CMSetColor bit is set in value mask.

num_colors

The number of color specifications in the color list. This value cannot exceed the Chart Manager constant MAX COLOR MODELS.

boundary

The boundaries of the map to be displayed. The MBoundary(3Map) man page discusses map display boundaries in detail. This specification is used only if the CMSetBoundary bit is set in value mask.

features

A list of features to be displayed. The MFeatAtts(3Map) man page discusses map features in detail. This specification is used only if the CMSet- Features bit is set in value mask.

num_features

The number of feature attributes in the feature list. There is no built in limit to the number of features which can be specified in an MChangeMap request.

The MError manual page provides more information on the fields in the MapErrorCodeInformation structure

ERRORS

Note 1: Errors returned by an internal change map request are similar, except that they are sent to the owner of the window.

Note 2: Where indicated, some of the errors below return a MapErrorCodeInformation structure. In the case of errors related to drawing features, the map type and sub type fields in this structure are set to the internal constant AnyMap. The counterpart feature fields are set to the feature which caused the error. Likewise, for errors related to drawing map products, the feature type and feature subtype fields are set to the internal constant Any- Feature. The counterpart map fields are set to the map product which caused the error. The MuReference(3Mu) calls can be used to obtain a string equivalent value for any of the fields.

AlreadyDrawingMap

A map draw command is already in progress for the specified window. This is a synchronous error in all cases.

BadChannel

An invalid channel id was used. This is a synchronous error in all cases.

BadMapEntry

An error occurred while drawing one of the map products. This error is due to problems with the map data format, missing data files, or other related problems. This is an asynchronous, non-fatal error. An associated MapErrorCodeInformation structure is returned with this error which gives specific information on the particular map product generating this error.

BadServer

The Draw Module responsible for drawing the map product has terminated. This is usually an asynchronous error. The MapErrorCodeInformation structure indicates the product generating the error.

BadWindowId

An invalid window id was used. This is a synchronous error in all cases.

BadValueError

An invalid value was specified in one or more of the MapChangeAttributes fields. This is a synchronous error in all cases.

ErrorDrawingMap

An error occurred on a specific draw request. This is an asynchronous, non-fatal error. An associated MapEr- rorCodeInformation structure is returned with this error which gives specific information on the particular map product generating this error.

ErrorDrawingFeature

An error occurred while a feature draw was in progress. This is an asynchronous, non-fatal error. An associated MapErrorCodeInformation structure is returned with this error which gives specific information on the particular map feature generating this error.

FeatureNotAvailable

Specified feature is not available. This is a synchronous, non-fatal error, which returns the feature generating the error in a MapErrorCodeInformation structure.

FeatureNotSupported

A specified feature is not supported on the given set of map products. This can occur due to restrictions inside a Draw Module for supporting the given feature under the current set of Draw Modules, display projection, earth model, and coverage. This is a synchronous, non-fatal error, which returns the map feature generating the error in a MapErrorCodeInformation structure.

HardwareNotSupported

A specified map could not be rendered because the underlying hardware does not support TrueColor. This is a synchronous, non-fatal error, which returns the map product generating the error in a MapErrorCodeIn- formation structure.

MapDrawAborted

The specified draw request was aborted at the request of some Chart Client. This is an asynchronous nonfatal error. The MapErrorCodeInformation structure is returned along with the error, and indicates the product generating the error.

MapTooSmall

The specified boundary results in an image which is too small to draw. This is a synchronous, non-fatal error. The MapErrorCodeInformation structure is returned along with the error, and indicates the product generating the error.

MaxExtents

The specified coverage inside the request exceeds some boundary extent in the given projection. The view is automatically recentered so that the extents are not exceeded. This is a non-fatal error. A MapErrorCodeIn- formation structure is returned along with the error, and indicates the projection generating the error.

MaxScale

The specified map scale exceeds the maximum scale supported. This error can occur under two conditions. In one case, the error refers to a scale factor which is projection dependent. In this case, the view is automatically set to the maximum scale. In the second case, the error refers to a specific map or feature product, which cannot be drawn because the view is zoomed out too far. In this case, a MapErrorCodeInfor- mation structure is returned along with the error. This is a non-fatal error in both cases.

MinScale

The specified map scale is smaller than the minimum scale supported. This error can occur under two conditions. In one case, the error refers to the desired coverage. In this case, the coverage is checked against an absolute factor common to all projections. The view is automatically set to the minimum scale. The second case occurs when a given map or feature product cannot be drawn because the scale is too small (eg. the view is zoomed too far in) for the map to be seen. The MapErrorCodeInformation structure is returned along with the error in this case. This is a non-fatal error in both cases.

NoMapsDrawn

The change map request resulted in no map products being drawable. This is a fatal error, which can be either synchronous or asynchronous. In either case, other non-fatal errors may precede it.

NotEnoughColors

The chart graphics server is unable to allocate the colors necessary to draw the indicated map. This is a non-fatal, asynchronous error. The MapErrorCodeInformation structure which is returned in this case describes the map product which could not be drawn.

OutOfMemory

Unable to allocate memory to perform request. This error can also occur if the image memory cannot be allocated for a particular product. In the latter case, an associated MapErrorCodeInformation structure is also returned.

ProductNotFound

The specified product was not found at the given area of coverage. This is a synchronous, non-fatal error, which returns a MapErrorCodeInformation structure indicating the product generating the error.

ProjectionNotSupported

The specified projection cannot display the given map or feature product. This is a synchronous, non-fatal error. The MapErrorCodeInformation structure indicates the product and projection generating the error. If a MapErrorCodeInformation structure is not included with this error, then the given projection is simply unsupported by the Chart Manager.

SystemNotSupported

The calculated earth model cannot display the given map or feature product. This is a synchronous, non-fatal error. The MapErrorCodeInformation structure indicates the product generating the error.

TooManyMaps

The change map request would result in more than MAX_MAPS draw requests if fully satisfied. This is a non-fatal, synchronous error. The net effect is to trim the number of draw requests to a maximum of MAX_MAPS.

UnresponsiveDrawModule

A Draw Module is not responding to a pending request. This is an asynchronous, non-fatal error. The MapErrorCodeInformation structure indicates the product generating the error.

WorldFitProblem

A view of the entire world is not possible in the given projection for the given pixmap and window extents. When drawing the whole world, the Chart Graphics server

attempts to fit a view of the world to either the window width or the window height boundary. The other boundary is extended as needed. If this extension goes beyond the extents of the pixmap, then the world view is clipped in that direction, and this warning is issued. This is a synchronous, non-fatal error. A

MapErrorCodeInformation structure is returned along with the error, and indicates the projection generating the error.

SEE ALSO

MAddFeature(3Map), MAddFeatures(3Map), MAddProduct(3Map), MAddProducts(3Map), MAttsMask(3Map), MBoundary(3Map), MColor(3Map), MFeatAtts(3Map), MListMaps(3Map), MModifyFeature(3Map), MModifyFeatures(3Map), MQueryMap(3Map), MProdAtts(3Map), MProjection(3Map), MRecenterMap(3C), MRemoveFeature(3Map), MRemoveFeatures(3Map), MRemoveProduct(3Map), MRemoveProducts(3Map), MSetMapColors(3C), MSetMapColorsByRGB(3C), MSetIntensity(3Map), MSetMapBounds(3Map), MSetMapWidth(3C), MScaleMap(3Map)

MChangeSymbol

FUNCTION

Modify the drawn symbol in a symbol object.

SYNTAX

C Interface

```
void MChangeSymbol( channel, object, symbol )
Channel   channel;
ObjectId  object;
NTDSSymbol     symbol;
```

ARGUMENTS

channel — The connection to the Chart Manager; returned from MOpenChannel.

object — The object whose symbol is to be changed.

symbol — The new symbol to be drawn.

DESCRIPTION

MChangeSymbol changes the NTDS symbol displayed in a symbol object, without the overhead of destroying and creating the object. Using this command on non-symbol objects will cause a BadValueError.

ERRORS

BadChannel
: An invalid channel id was used.

BadObjectId
: An invalid object id was used.

BadValueError
: The object is not a Symbol object.

SEE ALSO

MDrawSymbol(3Map), MSetObjData(3Map), MSetSymbSz(3Map)

FUTURE EXPANSIONS

The MChangeSymbol command is a simplified interface to the MSetObjectData() function.

MChangeText

FUNCTION

Modify the text field in a text object.

SYNTAX

C Interface

```
void MChangeText( channel, object, text, ntext ) Channel   channel;
                  ObjectId  object;
                  char *text;
                  int  ntext;
```

ARGUMENTS

channel — The connection to Cartographer; returned from MOpenChannel.

object — The object wholse text field is to be changed.

text — The new text string for the object.

ntext — The number of characters in the text string.

DESCRIPTION

MChangeText changes the text display in a Text object, without the overhead of destroying and creating the object. Using this command on objects whose type is not Text (including AngleText) will cause a BadValueError.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid object id was used.

BadValueError

The object is not a Text object.

SEE ALSO

MDrawText(3C), MSetObjectData(3C)

FUTURE EXPANSIONS

The MChangeText command is provided for compatability with earlier systems, and provides a simplified interface to the MSetObjectData() routine.

MChannelToSocket

FUNCTION

Return the socket id of the specified channel.

SYNTAX

C Interface

```
int MChannelToSocket( channel )
        Channel  channel;
```

ARGUMENTS

channel	The connection to Cartographer; returned from MOpenChannel.

DESCRIPTION

The MChannelToSocket function returns the socket id of the given channel, or (-1) if the specified channel is invalid. This function is useful if a Client application is to make a call to select(2). Many graphics toolkits allow sockets to be added as input sources; with this function you can get the socket id, and add it to eliminate the need to poll through the MPending(3C) call.

RETURN

The file descriptor for the socket is returned. The value -1 is returned in the case of an error.

ERRORS

BadChannel

An invalid channel id was used.

SEE ALSO

MOpenChannel(3C), MCloseChannel(3C)

MCloseChannel

FUNCTION

Close a communication channel to Cartographer.

SYNTAX

C Interface

```
void MCloseChannel( channel )
        Channel  channel;
```

ARGUMENTS

channel	The connection to Cartographer; returned by MOpen-Channel.

DESCRIPTION

MCloseChannel closes the communication channel between the application program and the Cartographer Manager specified by channel. All of the objects owned by the channel are removed from the Map. If this channel is the only owner of a window, the window will be destroyed.

ERRORS

BadChannel

The channel id was invalid.

SEE ALSO

MOpenChannel(3C)

MCopyTemplate

FUNCTION

Create a new template; take values from specified template.

SYNTAX

C Interface

```
ObjectId MCopyTemplate( channel, template)
Channel  channel;
ObjectId template;
```

ARGUMENTS

channel Specifies the connection to Cartographer; returned by MOpenChannel.

template Specifies the ID of the template to be copied.

DESCRIPTION

MCopyTemplate creates a new template whose ObjectAttributes are a copy of the attributes of template.

RETURNS

The ObjectId value of the new template. The value Invali- dObjectId is returned if the call fails.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid template id was used.

SEE ALSO

MCreateTemplate(3C)

MCreateClass

FUNCTION

Create an object class.

SYNTAX

C Interface

```
ObjectId MCreateClass( channel, window, atts,
    value mask, objects, nobjects )
        Channel  channel;
        WindowId window;
        ObjectAttributes   *atts;
        MapValueMask  value mask;

        ObjectId *objects;
            int nobjects;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned by MOpenChannel.

window — Specifies the window where the class of objects applies.

atts — Specifies a set of object attributes that applies to the class of objects.

value mask — A bit mask which represents fields in the atts structure which should be used.

objects — A list of objects which are the initial members of the class.

nobjects — The number of objects in the list.

DESCRIPTION

MCreateClass forms a Class of objects. An object can be a member of many Classes, and a Class can be used to set attributes (such as visibility) on groups of objects through the use of only one command. An object Class contains a list of objects and an associated ObjectAttributes structure. The objects contained in the Class list are made up of Lists and Drawables. The ObjectAttributes for the Class are initialized to atts. If atts is set to NULL, or for

those values of atts where the corresponding bit field in value mask is not set, the value is taken from the Defaul- tAttributes structure for that window.

RETURNS

The Object ID of the created object Class.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid object ID (one which is neither a List or a Drawable) was specified in the objects parameter.

BadWindowId

An invalid window id was used.

BadValueError

An invalid value has been specified in one of the ObjectAttributes fields.

SEE ALSO

MDestroyObject(3C) MAddObject(3C), MRemoveObject(3C)

MCreateList

FUNCTION

Create a list in a window.

SYNTAX

C Interface

```
ObjectId MCreateList( channel, window, objects, nob- jects )
        Channel         channel;
        WindowId  window;
        ObjectId  *objects;
        int  nobjects;
```

ARGUMENTS

channel	The connection to Cartographer; returned from MOpenChannel.
window	The window to creat the list in.
objects	The objects in the list.
nobjects	The number of objects in the list.

DESCRIPTION

The MCreateList function creates a list that will contain the objects passed. The number of ids may be 0, and the pointer to the array NULL. This will create a list with no children. Objects contained within the List must be Drawables or Lists of other objects. The object id of the newly created list is returned. If the list can not be created, InvalidObjectId is returned.

RETURNS

The ObjectId value of created object list. The value InvalidObjectId is returned if the call fails.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid object id was specified in the list.

BadWindowId

An invalid window id was used.

SEE ALSO

MDestroyList(3C), MAddObject(3C), MRemoveObject(3C)

MCreateObject

FUNCTION

Create an object through animation.

SYNTAX

C Interface

ObjectId MCreateObject(channel, template, object type) Channel channel;
ObjectId template;
ObjectType object type;

ARGUMENTS

channel Specifies the connection to Cartographer; returned from MOpenChannel.

template Specifies the ObjectAttributes used when the object is created.

object type
Specifies the type of Drawable object to be created.

DESCRIPTION

The MCreateObject function will create an object of the specified type using animation. Valid objects which can be created via animation include:

Arc
This function requires three points: a center point and the two endpoints of the arc.

Box
This function requires two points: a center point and a point on the box.

Circle
This function requires two points: a center point and a point on the circle.

Ellipse
This function requires three points: a center, a point to specify the major axis, and a point to specify the minor axis.

Line
This function requires two points to specify the line's

endpoints.

Polygon
This function requires N points for an N sided polygon. Animation extends from

the previous line segment and the polygon is created only after the finish key or third button on the input device is pressed. The MCreatePoly call allows Clients to place restrictions on the maximum number of points in the created object.

Polyline

This function requires N points for an N segment polyline. Animation extends from the previous line segment and the polyline is created only after the finish key or third button on the input device is pressed. The MCreatePoly call allows Clients to place restrictions on the maximum number of points in the object.

The create object call makes use of animate keys. The animate keys are defined by the latest MSetAnimateKeys call.

If the escape key is pressed anytime during the animate process, then the animation is aborted, and no object is created. If the select key is pressed during the animate process, this is equivalent to selecting a point. If the finish key is pressed and there are insufficient points to create an object definition, then this key is ignored. If enough points have been entered to complete an object's definition, then animation mode stops and the object is created. If the right input device button is pressed, this is equivalent to the finish key being pressed.

If an animation is already in effect, then a BadOwner error occurs.

Upon successful creation of an object, an ObjectChangedEvent is sent to the application which requested the creation. Also, the application which requested the creation is registered as the object's owner.

RETURNS

The ObjectId value of created object. The value InvalidOb- jectId is returned if the call fails. Note that the returned value is not an active object until a ObjectChanged event occurs for this object.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

An invalid window id was used.

BadObjectId

An invalid template id was used.

BadOwner

An animation is already in effect.

BadValueError

An invalid object type was specified.

SEE ALSO

MAbortAnimation(3C), MCreatePoly(3C), MCreateText(3C), MEvents(3C), MObjAtts(3C), MSetAnimateKeys(3C)

MCreatePoly

FUNCTION

Create a polygon or polyline object through animation.

SYNTAX

C Interface

```
ObjectId MCreatePoly( channel, template, object type, max points )
        Channel   channel;
        ObjectId  template;
        ObjectSubType  object type;

                int  max points;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

template — Specifies the ObjectAttributes used when the object is created.

object type — Specifies the type of drawable to be created, only Polygon or Polyline should be specified here. For other object types, use the conventional MCreateObject(3C) call.

max points — Specifies the number of points for a polygon or polyline.

DESCRIPTION

The MCreatePoly call creates a polygon or polyline object using animation. It differs from MCreateObject only because of an additional parameter, max points, which allows a Client to restrict the maximum number of points which an object can have. For more information, see MCreateObject.

RETURNS

The ObjectId value of created object. The value InvalidObjectId is returned if the call fails. Note that the returned value is not an active object until a ObjectChanged event occurs for this object.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId
An invalid window id was used.

BadObjectId
An invalid template id was used.

BadOwner
An animation is already in effect.

BadValueError
An invalid object type was specified. This can also occur if a bogus max points value is specified.

SEE ALSO
MAbortAnimation(3C), MCreateObject(3C), MEvents(3C), MObjAtts(3C), MSetAnimateKeys(3C)

MCreateTemplate

FUNCTION

Create a template.

SYNTAX

C Interface

```
ObjectId MCreateTemplate( channel, window, atts, value mask )
        Channel  channel;
        WindowId window;
        ObjectAttributes   *atts;
        MapValueMask  value mask;
```

ARGUMENTS

channel

Specifies the connection to Cartographer; return by MOpenChannel.

window

Specifies the window on which to create the template.

atts Specifies the object attributes to be assigned to this template. A value of NULL will create a template with all attribute values set to their default values.

value mask

Specifies a mask of object attributes used for setting the template's initial values. Object attributes which are not set take on their default values. Object masks are defined in MObjMask(3C).

DESCRIPTION

MCreateTemplate creates an ObjectAttributes template in the server. The values for the various fields are taken from the parameter atts if the corresponding bit in the value mask is set. Otherwise, the default value is used; see MObjAtts(3C) for a discussion of each attribute field and its default value. If atts is set to NULL, then a template with default attributes is created.

RETURNS

The ObjectId value of the created template. The value InvalidObjectId is returned if the call fails.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId
An invalid window id was used.

BadValueError
An invalid value in one or more fields of the ObjectAt- tributes structure occurred.

SEE ALSO
MApplyTemplate(3C), MCopyTemplate(3C), MObjAtts(3C), MObjMask(3C), MSetTemplate(3C)

MCreateText

FUNCTION

Place text on the screen through animation.

SYNTAX

C Interface

```
ObjectId MCreateText( channel, template, text, ntext ) Channel   channel;
                      ObjectId  template;
                      char *text;
                      int  ntext;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

template — Specifies the ObjectAttributes used for drawing the text.

text — The text string to be placed on the Map.

ntext — The number of characters in the string to be displayed.

DESCRIPTION

The MCreateText function allows the user to place text interactively on the map. Once the call is made, the text will appear on the screen, and the user can move it around with the cursor. When the left mouse button is pressed, a location for the text is selected, and a line is drawn from the location of the text to the current cursor location. The line can then be moved around to give it an offset from the location specified.

The MCreateText call makes use of animate keys. The animate keys are defined by the latest MSetAnimateKeys call.

If the escape key is pressed anytime during the animate process, then the animation is aborted, and no text is created. If the select key is pressed during the animate process, this is equivalent to selecting a point, and the text is created at the cursor location. The function of the finish key is similar in this case.

Upon successful creation of a text object, an ObjectChangedEvent is sent to the application which requested the creation. Also, the application which

requested the creation is registered as the object's owner. If an animation is already under

way, then a BadOwner error occurs.

RETURNS

The ObjectId value of created object. The value InvalidObjectId is returned if the call fails. Note that the returned value is not an active object until a ObjectChanged event occurs for this object.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

An invalid window id was used.

BadObjectId

An invalid template id was used.

BadOwner

An animation is already under way.

BadValueError

An invalid object type was specified.

SEE ALSO

MAbortAnimation(3C), MEvents(3C), MObjAtts(3C)

MCreateWindow

FUNCTION

Create a window in Chart showing world view.

SYNTAX

C Interface

WindowId MCreateWindow(channel, atts, atts mask, map, map mask)

```
Channel  channel;
WindowAttributes   *atts;
MapValueMask  atts mask;

MapChangeAttributes     *map; MapValueMask  map mask;
```

ARGUMENTS

channel The connection to Chart; returned by MOpenChannel.

atts Attributes that describe the window to be created.
See MWindowAtts(3C) for details about this structure.

atts mask For those attributes whose corresponding bit field is not set in atts mask, the default value is used.

map An initial map to be drawn. If set to NULL, then the default map will be drawn.

map mask A bit mask representing the attributes of the map to be modified.

DESCRIPTION

The MCreateWindow function creates a window with the specified attributes. A default map showing a vector map product at world view is drawn if no map attributes are specified. The window attributes structure contains a list of initial attributes for creating the window. See MWindowAtts(3C) for details. The atts mask field is further defined in MWinMask(3C). Default values for window attributes are also described in those sections.

RETURNS

The WindowId of the created window. The value

InvalidWindowId is returned if the call fails.

ERRORS

BadChannel

An invalid channel id was used.

BadDisplay

The window could not be created because the X Windows display specification does not exist or is inaccessible.

BadValueError

An invalid value was specified in one or more of the attributes.

DuplicateWindow

The request to create this window was aborted because the named window already exists.

TooManyWindows

The request to create this window was aborted because the system parameter for the number of windows supported by Chart was exceeded.

BUGS

The pixmap width and pixmap height fields should never be smaller than the width and height of the displayed window. Generally, Chart does not allow this to happen. However, in the event that the map window is reparented to another window, Chart loses control over the size of the map window. In the event that the specified pixmap width or pixmap height is then smaller than the map window, unpredictable results can occur.

SEE ALSO

MDestroyWindow (3C), MEventMask (3C), MQueryWindow (3C), MMapWindow (3C), MUnMapWindow (3C), MUseWindow (3C), MUseNamedWindow (3C), MWindowAtts (3C), MWinMask (3C), XOpenDisplay (Xlib)

MDebug

FUNCTION

The Chart Manager Debugging Utilities.

SYNTAX

C Interface

```
void MAllocVerify();
```

Several debugging utilities are available for debugging application programs that use the Chart Manager. C programmers who use MAlloc, MReAlloc, and MFree can compile their programs with the -DDEBUG_MALLOC compile switch in order to verify that their application is using proper memory management. When this switch is enabled, and a call is made to MAlloc() or MReAlloc(), a print statement occurs indicating the filename making the call, the line number of the call, and the resulting address of the allocated space.

MAllocVerify() can be called by your program at any time to retrieve current memory allocation statistics. This call is machine dependent, and has varying levels of effectiveness based on the machine which it is called from.

A number of other machine dependent debugging features are also available, including a memory allocation address value checker (eXT and Sun), and a memory allocation exception handler (NeXT). Refer to the malloc(3) manual page for available capabilities on your machine. If the M library is compiled with -DDEBUG_MALLOC, some of these machine dependent capabilities are automatically enabled.

MSetFPDebugLevel() allows the caller to specify actions to take during floating point exceptions. The effect of this call is machine dependent.

Available to both Ada and C programmers is a single step capability using the environment variable MapSingleStep. If this variable is set, then the channel between the application and the Chart Manager to which it is connected is flushed after every request. In addition, the call does not return until AFTER the request has been actually serviced by the Chart Manager. Normally, requests will be buffered and sent in a group. If an error occurs, it may be signaled well after the call is actually made.

ENVIRONMENT

MapSingleStep

If set using setenv(1), then application's communication with the Chart Manager enters a single step mode.

SYSTEM DEPENDENCIES

HP Series 300, 400, 700, 800,

MAllocVerify()calls memorymap(3). Otherwise, no special handling is performed.

Sun 3, Sun 4, SparcStations, DTC-2

MAllocVerify() calls malloc verify(3) to verify the memory heap, and prints a message if something is wrong. It also prints out a map of all allocated addresses by calling mallocmap(3).Programs MAlloc() also get the benefits of malloc debug(3), which is automatically set during the first call to MAlloc().

NOTES

None of the memory debugging capabilities are available unless both the tools library (libtools.a) and your program have been compiled with "DEBUG_MALLOC" defined as a C preprocessor variable. The option "-DDEBUG_MALLOC" works on the C compiler command line. Better yet, use imake, with the configuration files located in ${LIBS}/config, and set the "DebugMalloc" variable in the site.def file to the value "YES".

SEE ALSO

MMemory(3Map),

MDestroyList

FUNCTION

Destroy an object and its children.

SYNTAX

C Interface

```
void MDestroyList( channel, object )
        Channel  channel;
        ObjectId object;
```

ARGUMENTS

channel — The connection to Cartographer; returned by MOpenChannel.

object — The ID of the object to be destroyed.

DESCRIPTION

MDestroyList destroys the object from the object data base of the specified window. If the object is a list, all of the children of the list are destroyed recursively. If the object is not a list, this call is identical to MDestroyObject. If the object is visible within the Map Window, it will first be removed from the screen, and then destroyed. Any further references to this object ID will result in a BadObjectId error.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

The specified object does not exist.

SEE ALSO

MCreateList(3C), MDestroyObject(3C)

MDestroyObject

FUNCTION

Destroy an object in Cartographer.

SYNTAX

C Interface

```
void MDestroyObject( channel, object )
        Channel  channel;
        ObjectId object;
```

ARGUMENTS

channel Specifies the connection to Cartographer; returned by MOpenChannel.

object The ID of the object to be destroyed.

DESCRIPTION

MDestroyObject destroys the object from the object data base of the specified window. If the object is visible within the Map Window, it will first be removed from the screen, and then destroyed. If the object is a List or a Class, only the list or class will be destroyed, its children will remain unchanged. Any further references to this object ID will result in a BadObjectId error.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

The object specified does not exist.

SEE ALSO

MDestroyList(3C)

MDestroyWindow

FUNCTION

Destroy a window in Cartographer.

SYNTAX

C Interface

```
void MDestroyWindow( channel, window )
        Channel  channel;
        WindowId window;
```

ARGUMENTS

channel	Specifies the connection to Cartographer; returned by MOpenChannel.
window	Specifies the window to be destroyed.

DESCRIPTION

MDestroyWindow destroys the window in the server. All data associated with the window is freed within the server. Any further reference to window will result in a BadWindowId error. If the window is currently mapped, it will be unmapped before it is destroyed.

The window's owner is defined to be the Client which created the window. The window is destroyed ONLY IF the specified Client is the window's owner. Otherwise, the window connection is released, and the window is destroyed ONLY IF this Client is the last remaining connection to the window.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

The window id passed is not valid.

BadOwner

The specified Client is not connected to the indicated window.

SEE ALSO

MCreateMapWindow(3C), MMapWindow(3C), MUnMapWindow(3C), MUseWindow(3C), MReleaseWindow(3C)

MDrawArc

FUNCTION

Draw an arc to a window.

SYNTAX

C Interface

```
ObjectId MDrawArc( channel, template, center, bearing, major axis, minor axis,
angle1, angle2 )
        Channel     channel;
        ObjectId    template;
        MapPoint    *center;
        FLOAT       bearing;
        FLOAT       major axis;

        FLOAT       minor axis;
        FLOAT       angle1;
        FLOAT       angle2;
```

ARGUMENTS

channel The connection to Cartographer; returned from MOpenChannel.

template A reference to ObjectAttributes used for drawing the arc.

center The center point of the arc, in decimal degrees.

bearing The bearing of the major axis from true north in degrees.

major axis
The major axis of the arc, in nautical miles.

minor axis
The minor axis of the arc, in nautical miles.

angle1 The start of the arc in degrees from true north.

angle2 The extent of the arc in degrees from true north.

DESCRIPTION

The MDrawArc function draws a circular or elliptical arc to the specified window. Each arc is specified by a center

point and two angles. The arc will be drawn around this center point with the given major and minor axes.

The template may be a Template, Class, or List object. In any case, the attributes from the template object are copied into the attributes of the newly created object. If the template is a List or a Class, the newly created object is added as a member.

RETURNS

The ObjectId value of the created object. The value InvalidObjectId is returned if the call fails.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid template id was used.

BadValueError

An invalid value for one of the arc parameters was specified.

SEE ALSO

MObjAtts(3C), MDestroyObject(3C)

MDrawBitmap

FUNCTION

Draw a bitmap.

SYNTAX

C Interface

ObjectId MDrawBitmap(channel, template, location, data, width, height, x hot, y hot)

```
Channel     channel;
ObjectId    template;
MapPoint    *location;
char        *data;
int         width;
int         height;
int         x hot;

int         y hot;
```

ARGUMENTS

channel Specifies the connection to Cartographer; returned from MOpenChannel.

template Specifies the ObjectAttributes used for drawing the bitmap.

location The location of the bitmap on the map.

data The bitmap data, in X bitmap format.

width The width of the bitmap in pixels.

height The height of the bitmap in pixels.

x hot The x origin of the bitmap in pixels.

y hot The y origin of the bitmap in pixels.

DESCRIPTION

The MDrawBitmap function draws a bitmap in the specified window. The bitmap is drawn at the location specified, and is centered around the point within the bitmap specified by (x_hot, y_hot). The data must be in X Bitmap format.

The template may be a Template, Class, or List object. In any case, the attributes from the template object are copied into the attributes of the newly created object. If the template is a List or a Class, the newly created object is added as a member.

RETURNS

The ObjectId value of the created object. The value Invali- dObjectId is returned if the call fails.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid template id was used.

BadValueError

An invalid bitmap, location, or hot spot was specified.

SEE ALSO

MDestroyObject(3C), MObjAtts(3C)

MDrawBox

FUNCTION

Draw a box.

SYNTAX

C Interface

```
ObjectId MDrawBox( channel, template, center, bearing, width, height )
        Channel     channel;
        ObjectId    template;
        MapPoint    *center;
        FLOAT       bearing;
        FLOAT       width;
        FLOAT       height;
```

ARGUMENTS

channel Specifies the connection to Cartographer; returned from MOpenChannel.

template Specifies the ObjectAttributes used for drawing the box.

center Specifies the center point of the box.

bearing Specifies the bearing of the box to be drawn.

width Specifies the width of the box in nautical miles.

height Specifies the height of the box in nautical miles.

DESCRIPTION

The MDrawBox function draws a box in the specified window. The box is centered about the specified center point, with the appropriate width and height. This object differs from a rectangle in that it can be moved about the center point with the MMoveObject function.

The template may be a Template, Class, or List object. In any case, the attributes from the template object are copied into the attributes of the newly created object. If the template is a List or a Class, the newly created object is added as a member.

RETURNS

The ObjectId value of the created object. The value Invali- dObjectId is returned if the call fails.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId
An invalid template id was used.

BadValueError
An invalid bearing, center, width, or height was specified.

SEE ALSO
MObjAtts(3C), MDestroyObject(3C)

MDrawChar

FUNCTION

Draw a single character.

SYNTAX

C Interface

```
ObjectId MDrawChar( channel, template, character, loca- tion, x offset, y offset )
    Channel     channel;
    ObjectId    template;
    int         character;
    MapPoint    *location;
    int         x offset;

                int         y offset;
```

ARGUMENTS

channel Specifies the connection to Cartographer; returned from MOpenChannel.

template Specifies the ObjectAttributes used for drawing the symbol.

character The character to be drawn.

location The center point of the character to be drawn.

x offset The x pixel offset from the location for the character.

y offset The y pixel offset from the location for the character.

DESCRIPTION

The MDrawChar function draws a single character to the specified window using the font and color specified by tem- plate. The character is drawn offset from location, the offset is specified by the parameters x offset, and y offset. This command should NOT be used for drawing the NTDS symbol fonts defined in Appendix A because those fonts are 16 bit fonts. The MDrawChar16 or MDrawSymbol calls should be used instead.

The template may be a Template, Class, or List object. In any case, the attributes from the template object are copied into the attributes of the newly created object. If the template is a List or a Class, the newly created object is added as a

member.

RETURNS

The ObjectId value of the created object. The value Invali- dObjectId is returned if the call fails.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid template id was used.`

BadValueError

An invalid location was specified.

SEE ALSO

MDestroyObject(3C), MDrawChar16(3C), MDrawSymbol(3C), MObjAtts(3C), MSetOffset(3C)

MDrawChar16

FUNCTION

Draw a single character.

SYNTAX

C Interface

```
ObjectId MDrawChar16( channel, template, character, location, x offset, y offset )
        Channel         channel;
        ObjectId  template;
        short           character;
        MapPoint  *location;
        int  x offset;

        int  y offset;
```

ARGUMENTS

channel Specifies the connection to Cartographer; returned from MOpenChannel.

template Specifies the ObjectAttributes used for drawing the symbol.

character The character to be drawn.

location The center point of the character to be drawn.

x offset The x pixel offset from the location for the character.

y offset The y pixel offset from the location for the character.

DESCRIPTION

The MDrawChar16 function draws a single 16 bit character in the specified window using the font and color specified by template. The character is drawn at location. Certain special fonts, defined in appendix A, enable NTDS symbols to be drawn in various sizes and intensity. Integer constants are available for drawing these symbols. A symbol's font can be changed dynamically by updating the ObjectAttributes font name field after creating the symbol. Refer to the FONT INFORMATION section under the MDrawSymbol(3C) manual page for more information.

The only difference between this and MDrawChar is that this routine provides access to fonts with more than 256 symbols.

The template may be a Template, Class, or List object. In any case, the attributes from the template object are copied into the attributes of the newly created object. If the template is a List or a Class, the newly created object is added as a member.

RETURNS

The ObjectId value of the created object. The value Invali- dObjectId is returned if the call fails.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid template id was used.`

BadValueError

An invalid location was specified.

SEE ALSO

MDestroyObject(3C), MDrawChar(3C), MDrawSymbol(3C), MObjAtts(3C), MSetOffset(3C)

MDrawCircle

FUNCTION

Draw a circle.

SYNTAX

C Interface

```
ObjectId MDrawCircle( channel, template, center, radius )
    Channel     channel;
    ObjectId    template;
    MapPoint    *center;
    FLOAT       radius;
```

ARGUMENTS

channel Specifies the connection to Cartographer; returned from MOpenChannel.

template Specifies the ObjectAttributes used for drawing the circle.

center Specifies the center point of the circle.

radius The radius in Nautical Miles of the circle.

DESCRIPTION

MDrawCircle draws a circle in the specified window. It uses the following object attributes to determine how the circle is to be passed: fill_style, line_style, color, and line_width.

The template may be a Template, Class, or List object. In any case, the attributes from the template object are copied into the attributes of the newly created object. If the template is a List or a Class, the newly created object is added as a member.

RETURNS

The ObjectId value of the created object. The value InvaliddObjectId is returned if the call fails.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid template id was used.

BadValueError

An invalid center or radius was specified.

SEE ALSO

MObjAtts(3C), MDestroyObject(3C)

MDrawEllipse

FUNCTION

Draw an ellipse.

SYNTAX

C Interface

ObjectId MDrawEllipse(channel, template, center, bear- ing, major axis, minor axis)

```
Channel     channel;
ObjectId    template;
MapPoint    *center;
FLOAT       bearing;
FLOAT       major axis;

FLOAT       minor axis;
```

ARGUMENTS

channel Specifies the connection to Cartographer; returned from MOpenChannel.

template Specifies the ObjectAttributes used for drawing the ellipse.

center Specifies the center point for the ellipse.

bearing The bearing of the ellipse in degrees.

major axis

The major axis of the ellipse in nautical miles.

minor axis

The minor axis of the ellipse in nautical miles.

DESCRIPTION

MDrawEllipse function draws an ellipse in the specified window. The ellipse is drawn with the specified major and minor axes. These values are in nautical miles. The major axis of the ellipse always runs along the line of bearing. The bearing is a measure in degrees from True North.

The template may be a Template, Class, or List object. In any case, the attributes from the template object are copied into the attributes of the newly created object. If the template is a List or a Class, the newly created object is

added as a member.

RETURNS

The ObjectId value of the created object. The value Invali- dObjectId is returned if the call fails.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid template id was used.

BadValueError

An invalid center, bearing, major axis, or minor axis was specified.

SEE ALSO

MObjAtts(3C), MDestroyObject(3C)

MDrawLine

FUNCTION

Draw a line.

SYNTAX

C Interface

ObjectId MDrawLine(channel, template, point1, point2) Channel channel;
ObjectId template;
MapPoint *p1;
MapPoint *p2;

ARGUMENTS

channel The connection to Cartographer; returned from MOpenChannel.

template Specifies the ObjectAttributes used for drawing the line segment.

p1 An endpoint for placing the line segment.

p2 An endpoint for placing the line segment.

DESCRIPTION

The MDrawLine function draws a line segment in the specified window.

The template may be a Template, Class, or List object. In any case, the attributes from the template object are copied into the attributes of the newly created object. If the template is a List or a Class, the newly created object is added as a member.

RETURNS

The ObjectId value of the created object. The value InvalidObjectId is returned if the call fails.

ERRORS

BadChannel
An invalid channel id was used.

BadObjectId
An invalid template id was used.

SEE ALSO

MObjAtts(3C), MDestroyObject(3C)

MDrawPolyLine

FUNCTION

Draw a multiple segment line.

SYNTAX

C Interface

```
ObjectId MDrawPolyLine( channel, template, points, npoints )
    Channel   channel;
    ObjectId  template;
    MapPoint  *points;
    int  npoints;
```

ARGUMENTS

channel Specifies the connection to Cartographer; returned from MOpenChannel.

template Specifies the ObjectAttributes used for drawing the line.

points A pointer to an array of points.

npoints The number of points in the array.

DESCRIPTION

The MDrawPolyLine function draws a polyline in the specified window.

The template may be a Template, Class, or List object. In any case, the attributes from the template object are copied into the attributes of the newly created object. If the template is a List or a Class, the newly created object is added as a member.

RETURNS

The ObjectId value of the created object. The value Invali- dObjectId is returned if the call fails.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid template id was used.

BadValueError

An invalid point was specified.

SEE ALSO

MObjAtts(3C), MDestroyObject(3C),

MDrawPolygon

FUNCTION

Draw a polygon.

SYNTAX

C Interface

```
ObjectId MDrawPolygon( channel, template, points, npoints )
    Channel   channel;
    ObjectId  template;
    MapPoint  *points;
    int  npoints;
```

ARGUMENTS

channel — The connection to Cartographer; returned from MOpenChannel.

template — Specifies the ObjectAttributes used for drawing the polygon.

points — A pointer to an array of points.

npoints — The number of points in the array.

DESCRIPTION

The MDrawPolygon function draws a polygon in the specified window. This routine works the same as MDrawPolyline except that it closes the polygon by adding the first point to the end of the polygon, and that a Polygon can be filled.

The template may be a Template, Class, or List object. In any case, the attributes from the template object are copied into the attributes of the newly created object. If the template is a List or a Class, the newly created object is added as a member.

RETURNS

The ObjectId value of the created object. The value InvaliddObjectId is returned if the call fails.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid template id was used.

BadValueError

An invalid point was specified.

SEE ALSO

MObjAtts(3C), MDestroyObject(3C),

MDrawRectangle

FUNCTION

Draw a rectangle.

SYNTAX

C Interface

```
ObjectId MDrawRectangle( channel, template, p1, p2 ) Channel   channel;
        ObjectId  template;
        MapPoint  *p1;
        MapPoint  *p2;
```

ARGUMENTS

channel — The connection to Cartographer; returned from MOpenChannel.

template — Specifies the ObjectAttributes used for drawing the rectangle.

p1 — The upper left hand corner of the rectangle.

p2 — The lower right hand corner of the rectangle.

DESCRIPTION

The MDrawRectangle function draws a rectangle in the specified window. The function will be drawn according to the line_style field of the ObjectAttributes structure. The line can be drawn as one of three types: GreatCircle, RhumbLine, or GeoDesic. The default line type is GeoDesic, which connects the two points with a straight line on the screen.

The template may be a Template, Class, or List object. In any case, the attributes from the template object are copied into the attributes of the newly created object. If the template is a List or a Class, the newly created object is added as a member.

RETURNS

The ObjectId value of the created object. The value InvalidObjectId is returned if the call fails.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid template id was used.

BadValueError
An invalid point was specified.

SEE ALSO
MObjAtts(3C), MDestroyObject(3C)

MDrawSector

FUNCTION

Draw a sector.

SYNTAX

C Interface

ObjectId MDrawSector(channel, template, center, bear- ing, range1, range2, angle1, angle2)

```
Channel     channel;
ObjectId    template;
MapPoint    *center;
FLOAT       bearing;
FLOAT       range1;
FLOAT       range2;
FLOAT       angle1;
FLOAT       angle2;
```

ARGUMENTS

channel The connection to Cartographer; returned from MOpenChannel.

template A reference to ObjectAttributes used for drawing the Sector.

center The center point of the arc, in decimal degrees.

bearing The bearing of the major axis from true north in degrees.

range1 The radius of the first arc in nautical miles.

range2 The radius of second arc in nautical miles.

angle1 The start of the arc in degrees from true north.

angle2 The extent of the arc in degrees from true north.

DESCRIPTION

The MDrawSector function draws two circular arcs connected by line segments. Both arcs are drawn with the same center point, are drawn through the same angular range, and have

radii of range1 and range2. The two arcs are connected by line segements.

The template may be a Template, Class, or List object. In any case, the attributes from the template object are copied into the attributes of the newly created object. If the template is a List or a Class, the newly created object is added as a member.

RETURNS

The ObjectId value of the created object. The value Invali- dObjectId is returned if the call fails.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid template id was used.

BadValueError

An invalid value for one of the arc parameters was specified.

SEE ALSO

MObjAtts(3C), MDestroyObject(3C)

MDrawSegment

FUNCTION

Draw a segment.

SYNTAX

C Interface

```
ObjectId MDrawSegment( channel, template, location, bearing, length )
    Channel     channel;
    ObjectId    template;
    MapPoint    *location;
    FLOAT       bearing;
    int         length;
```

ARGUMENTS

channel — The connection to Cartographer; returned from MOpenChannel.

template — Specifies the ObjectAttributes used for drawing the line segment.

location — The location to place the segment. This location specifies the starting point for the line segment to be drawn.

bearing — The bearing of the segment in degrees from true north.

length — The length of the segment in pixels.

DESCRIPTION

The MDrawSegment function draws a segment in the specified window. A segment is a line with a fixed length. The length is specified in pixels, not nautical miles. This causes the segment to remain a fixed size regardless of the scale of the map. The segment is drawn based on a bearing in degress from North. The length and bearing of a segment can be changed without having to destroy and recreate it using the function MSetSegment. The MDrawSlash call is similar, with only subtle differences.

The template may be a Template, Class, or List object. In any case, the attributes from the template object are copied into the attributes of the newly created object. If the template is a List or a Class, the newly created object is

added as a member.

RETURNS

The ObjectId value of the created object. The value Invali- dObjectId is returned if the call fails.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid template id was used.

BadValueError

An invalid location, bearing, or range was specified.

SEE ALSO

MObjAtts(3C), MDestroyObject(3C), MDrawSlash(3C), MSetOffset(3C), MSetSegment(3C)

MDrawSlash

FUNCTION

Draw a slash.

SYNTAX

C Interface

```
ObjectId MDrawSlash( channel, template, location, bear- ing, length )
        Channel     channel;
        ObjectId    template;
        MapPoint    *location;
        FLOAT       bearing;
        int         length;
```

ARGUMENTS

channel The connection to Cartographer; returned from MOpenChannel.

template Specifies the ObjectAttributes used for drawing the slash.

location The location to place the slash; the drawn line will be centered around this point.

bearing The bearing of the slash in degrees from true north.

length The length of the slash in pixels.

DESCRIPTION

The MDrawSlash function draws a slash in the specified window. A slash is a line with a fixed length. The length is specified in pixels, not nautical miles. This causes the slash to remain a fixed size regardless of the scale of the map. The slash is drawn based on a bearing in degress from North. Slash segments differ from line segments drawn by the MDrawSegment call in the following ways:

(1). The slash line is drawn with the specified location as the line's center point. The segment line is draw with the specified location as the line's starting point.

(2). A slash line cannot be moved using the MSetSegment call.

The template may be a Template, Class, or List object. In any case, the attributes from the template object are copied into the attributes of the newly created object. If the template

is a List or a Class, the newly created object is added as a member.

RETURNS

The ObjectId value of the created object. The value Invali- dObjectId is returned if the call fails.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid template id was used.

BadValueError

An invalid location, bearing, or range was specified.

SEE ALSO

MDestroyObject(3C), MDrawSegment(3C), MObjAtts(3C), MSetOffset(3C)

MDrawSymbol

FUNCTION

Draw a symbol.

SYNTAX

C Interface

```
#include <M/Symbols.h>
ObjectId MDrawSymbol( channel, template, symbol, size, location )
        Channel         channel;
        ObjectId  template;
        NTDSSymbol      symbol;
        int  size;
        MapPoint  *location;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

template — Specifies the ObjectAttributes used for drawing the symbol.

symbol — The NTDS symbol to be drawn.

size — The NTDS symbol font size to be drawn. Valid values are: Tiny, Small, Medium, Large, Huge, TinyBold, SmallBold, MediumBold, LargeBold, Huge- Bold.

location — The center point of the symbol to be drawn.

DESCRIPTION

The MDrawSymbol function draws a symbol in the specified window. This routine draws a symbol centered about the specified center point. The symbol must be one of the elements of the NTDS font. Most symbols in this font have predefined constants for the characters to add to a program's readability. The list of constants are referenced by including the file <M/Symbols.h>.

The symbols to be drawn can be of different sizes by modifying the size parameter. The template will affect the color of the symbol that is drawn; however, the font used by this

command is preselected (ie. the template font attribute is ignored). The MDrawChar() and MDrawChar16() functions are more general purpose.

The template may be a Template, Class, or List object. In any case, the attributes from the template object are copied into the attributes of the newly created object. If the template is a List or a Class, the newly created object is added as a member.

FONT INFORMATION

Cartographer Manager provides a set of symbols for drawing symbolic information onto the map. These symbols are located in defined 16 bit character fonts, and can be drawn using either the MDrawSymbol() or the MDrawChar16() calls. The predefined constants used by the MDrawSymbol() call map out to the following font names:

Tiny
-chart-ntds-medium-r-normal--13-130-7575-m-130-iso8859-1

Small
-chart-ntds-medium-r-normal--21-210-7575-m-210-iso8859-1

Medium
-chart-ntds-medium-r-normal--31-310-7575-m-310-iso8859-1

Large
-chart-ntds-medium-r-normal--41-410-7575-m-410-iso8859-1

Huge
-chart-ntds-medium-r-normal--51-510-7575-m-510-iso8859-1

TinyBold -chart-ntds-bold-r-normal--13-130-75-75-m-130-iso8859-1

SmallBold -chart-ntds-bold-r-normal--21-210-75-75-m-210-iso8859-1

MediumBold -chart-ntds-bold-r-normal--31-310-75-75-m-310-iso8859-1

LargeBold -chart-ntds-bold-r-normal--41-410-75-75-m-410-iso8859-1

HugeBold -chart-ntds-bold-r-normal--51-510-75-75-m-510-iso8859-1

Note: these fonts must be installed prior to attempting to use them. In the event that the font cannot be loaded, a substitute (text) font is used instead.

RETURNS

The ObjectId value of the created object. The value Invali- dObjectId is returned if the call fails.

STRUCTURES

C Interface

The NTDSSymbol construct is used for specifying predefined symbols from the NTDS font sets. Each NTDS character has a unique identifier constant defined in

the include file <M/Symbols.h>.

typedef int NTDSSymbol;

BadChannel
An invalid channel id was used.

BadObjectId
An invalid template id was used.

BadValueError
An invalid symbol, size, or location was specified.

SEE ALSO
MChangeSymbol(3C), MDestroyObject(3C), MObjAtts(3C), MSetOffset(3C), MSetSymbolSize(3C)

MDrawText

FUNCTION

Draw text.

SYNTAX

C Interface

```
ObjectId MDrawText( channel, template, text, ntext, location, x offset, y offset )
        Channel         channel;
        ObjectId  template;
        char *text;
        int  ntext;
        MapPoint  *location;
        int  x offset;

                                int  y offset;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

template — Specifies the ObjectAttributes used for drawing the text string.

text — Specifies the text to be drawn.

ntext — The number of characters in the string to be drawn.

location — Specifies the location of the text to be placed in the window.

x offset — The offset in pixels from the center point for the text.

y offset — The offset in pixels from the center point for the text.

DESCRIPTION

The MDrawText function draws a text string in the specified window. The text will be drawn with the font specified in its attributes. The text is positioned at the pixel offset specified by x offset and y offset, from the center point specified.

The template may be a Template, Class, or List object. In any case, the attributes from the template object are copied into the attributes of the newly created object. If the template is a List or a Class, the newly created object is added as a member.

RETURNS

The ObjectId value of the created object. The value Invali- dObjectId is returned if the call fails.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid template id was used.

BadValueError

An invalid location or offset was specified.

SEE ALSO

MChangeText(3C), MDrawAngleText(3C), MDestroyObject(3C), MObjAtts(3C), MSetOffset(3C)

MDrawWeather

FUNCTION

Draw a weather segment.

SYNTAX

C Interface

```
ObjectId MDrawWeather( channel, template, front type, points, npoints )
    Channel   channel;
    ObjectId  template;
    FrontType front type;

    MapPoint  *points;
    int  npoints;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

template — Specifies the ObjectAttributes used for drawing the line.

front type — Specifies what type of front is to be drawn.

points — A pointer to an array of points.

npoints — The number of points in the array.

DESCRIPTION

The MDrawWeather function draws a weather line to the specified window. Valid FrontTypes are: WarmFront, ColdFront, OccludedFront, and StationaryFront.

The template may be a Template, Class, or List object. In any case, the attributes from the template object are copied into the attributes of the newly created object. If the template is a List or a Class, the newly created object is added as a member.

RETURNS

The ObjectId value of the created object. The value Invali- dObjectId is returned if the call fails.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid template id was used.

BadValueError

An invalid point was specified.

SEE ALSO

MObjAtts(3C), MDestroyObject(3C),

MDrawWorld

FUNCTION

Draw a world view map in specified window.

SYNTAX

C Interface

```
void MDrawWorld( channel, window )
        Channel   channel;
        WindowId  window;
```

ARGUMENTS

channel	The connection to Cartographer; returned from MOpenChannel.
window	The window to which the map draw is directed.

DESCRIPTION

MDrawWorld draws a world view map to the given window. The projection, feature, and color models remain unchanged. The product list is based on the current set of map products, modified by setting the subtype fields to AnyMap. This allows products in the given class which are displayable at world view to be loaded in. If no products can be drawn from the current set, then the default set (VectorMap class, AnyMap subtype) is displayed.

ERRORS

SEE ALSO

n/a

MError

FUNCTION

Cartographer Manager error handling routines.

SYNTAX

C Interface

```
#include <M/Merror.h>
void MSetErrorHandler( handler)

        MapErrorProc    handler;

                void MResetErrorHandler( )

void MSetIOErrorHandler( handler) MapIOErrorProc    handler;

                void MResetIOErrorHandler( )

char *MErrorToString( error code) MapStatus    error code;

#include <M/Mproto.h>

char *MMajorCodeToString( major code) Protocol    major code;

char *MMinorCodeToString( minor code) Protocol    minor code;
```

ARGUMENTS

handler — An application specific error handler.

error code — The code number of the generated error. Error codes are described in each Chart Manager manual page under the "ERRORS" heading.

major code — The major code number of the generated error.

minor code — The minor code code number of the generated error.

DESCRIPTION

The Chart Manager Library has two asynchronous error handler routines that are called whenever an error occurs within the Chart Manager. One handler deals exclusively with IO errors and the other handles all other errors. Both error handlers can be replaced with user-defined handlers by using the routines MSetIOErrorHandler and MSetErrorHandler. Once the error handler is replaced by a user routine, the user routine will be called

whenever an error occurs. The default handlers that the library defines can be restored with the

routines: MResetIOErrorHandler and MResetErrorHandler. The Chart Manager error service provides a default handler which prints a message to the standard error device for all non-IO errors. This handler is also called if your client's error handler returns a non-zero value. The application can substitute its own routine in lieu of the default routine by using the MSetErrorHandler routine. Available to the user are three routines for converting internal Chart Manager codes to strings. These routines are: MErrorToString, MMa- jorCodeToString, and MMinorCodeToString.

The MErrorToString routine provides a small text description for each error code, the MMajorCodeToString routine provides a text description for each major protocol code in the Chart Manager, and the MMinorCodeToString routine provides a text description for each minor protocol code in the Chart Manager. All three routines return the text string "Unknown" in the case where the input code is not defined by the Chart Manager. All three routines return pointers to static string buffers which should not be modified by the caller.

The application error handler routine is called with the following format whenever a Chart Manager error occurs:

```
int
        (*handler) ( channel, window, major code, minor code, error code,
        error info)
                Channel         channel;
                WindowId        window;
                Protocol        major code;

                Protocol        minor code;
                MapStatus  error code; MapErrorCodeInformation   *error info;
```

The channel indicates the channel over which the error occurred. channel is set to InvalidChannel if the call generating the error has no associated direct communications with the Chart Manager.

The window parameter indicates the map window generating the error. window is set to InvalidWindowId if the call generating the error has no associated map window. Examples of such calls include MOpenChannel.

The major code and minor code can be used by the application to handle errors for a specific Chart Manager request. The error code parameter can be used by the application to handle specific errors. The error info is returned along with some errors (specifically

map related errors), to specify

additional information concerning the error. If the passed argument is non-NULL, then the information takes the form of a MapErrorCodeInformation structure (specified in the STRUCTURES section below).

The MResetErrorHandler call resets the error handler back to the default error handler.

The IO error handler prints the same information as the default error handler, it also causes the program to exit after ther message is printed. An IO error will be fatal and must be handled by the Client program in order for execution to continue. There parameters to the IO error handler are the same except that there is no window parameter.

RETURN

Certain error routines return a string value. This value is statically assigned, and should NOT need to be freed using free. Your library error handler, specified using the MSetErrorHandler() call, should return the value 0 if the error was processed, and the value non-zero if it was not. In the case where the error is not processed, the default error handler is called, and an error message is printed to the standard error (stderr) device.

STRUCTURES

C Interface

```
typedef int             MapStatus;

typedef short  Protocol;

typedef int             (*MapErrorProc)(
        Channel         channel,
        WindowId        window,
        Protocol        major_code,
        Protocol        minor_code,
        MapStatus  error_code,
MapErrorCodeInformation *error_info);

typedef void            (*MapIOErrorProc)(
        Channel         channel,
        Protocol        major_code,
        Protocol        minor_code,
                        MapStatus  error_code);

typedef struct {
        MapType         map_type;
                        MapSubType sub_type;
        FeatureType     feature_type; FeatureSubType  feature_subtype;
        ProjectionType  projection;
```

} MapErrorCodeInformation;

The Protocol value is used to uniquely determine the M library request generating the error. The MapStatus value classifies an error. The VALUES section of this manual page describes all possible values for MapStatus. The MapErrorProc is the name of a procedure which gets called if an error occurs. The MapIOErrorProc is the name of a procedure which gets called if an I/O error occurs. Note that both procedures have parameter definitions included if your C compiler supports this.

The MapErrorCodeInformation structure provides further information for certain kinds of errors. The fields in this structure are defined as follows:

map_type
: This field indicates the type of map product causing the error. These values are internally encoded. Values depend upon the Draw Modules currently connected to the Chart Manager. This field is set to the value AnyMap if the cause of the error is due to a feature product, and not a map product.

sub_type
: This field indicates the subtype of map product causing the error, given a specific map type. Similar to the map type field, this field is internally encoded.

projection
: This field indicates the projection causing the error. This field is internally encoded. This field specifies the name of the unsupported projection, for example, when a ProjectionNotSupported error occurs.

feature_type
: This field indicates the type of feature product causing the error. These values are internally encoded. Values depend upon the Draw Modules currently connected to the Chart Manager. This field is set to the value AnyFeature if the cause of the error is due to a map product, and not a feature product.

feature_subtype
: This field indicates the subtype of feature product causing the error, given a specific feature type. Similar to the feature type field, this field is internally encoded.

VALUES

MapStatus

AlreadyConnected
: The connection to the specified Chart Manager already exists. See MOpenChannel(3C).

AlreadyDrawingMap

A map draw request for the specified window is already in progress.

BadChannel

The specified channel is invalid.

BadDisplay

The referenced display is no longer valid. This can occur if a remote X server exits while Chart Manager has a window to it still defined.

BadMapEntry

BadObjectId

The specified object is invalid, or does not exist.

BadOwner

The specified application is attempting to use a map window without having previously performed a MUseWindow or MUseNamedWindow call.

BadRecord

The specified record in a particular file is invalid.

BadServer

A request is being made to draw a map whose responsible Draw Module has terminated.

BadTemplate

The specified template is invalid, or no longer exists.

BadValueError

The specified value is invalid for the given call.

BadVolume

The specified volume path does not exist, or else it cannot be read.

BadWindowId

The specified window is invalid, or has been previously deleted.

BadWindowName

The specified window name does not exist, or else

is invalid. See MUseNamedW(3Map).

DataSyncError

The Client/Manager protocol is out of sync. Some data may have been lost.

ErrorDrawingFeature

ErrorDrawingMap

FeatureNotAvailable
FeatureNotSupported
HardwareNotSupported
MapDrawAborted
MapNotFound
MapTooSmall
MaxExtents
MaxScale
MinScale
NoError
No error defined.

NoMapsDrawn

NotEnoughColors

OutOfMemory
The Chart Manager has run out of memory. This usually occurs because of a failure in the mal- loc(3) call.

ProductNotFound

ProjectionNotSupported

SocketError
An exception occurred on the specified socket. This is generally the error code seen by an M library I/O handler.
SystemNotSupported

TooManyEvents
The Chart Manager event queue is full. The event which caused this had to be thrown away.

TooManyMaps

UnresponsiveDrawModule

UnknownError
The specified error is undefined.

WorldFitProblem

SEE ALSO
MOpenChannel(3C), MUseWindow(3C), MUseNamedWindow(3C), malloc(3), stdio(3),

ERRORS section under each application call.

MExchangeObject

FUNCTION

Exchange an object between one class or list and another class or list.

SYNTAX

C Interface

void MExchangeObject(channel, object, oldclass, newclass, atts mask)

```
Channel         channel;
                ObjectId  object;
                ObjectId  oldclass;
                ObjectId  newclass;
MapValueMask    atts mask;
```

ARGUMENTS

channel The connection to Cartographer; returned from MOpenChannel.

object The object being moved from one class or list to another class or list.

oldclass The list or class from which the object will be removed.

newclass The list or class to which the object will be added.

atts mask The object attributes mask to apply when applying the newclass object attributes to object.

DESCRIPTION

MExchangeObject moves the specified object from one class or list to another class or list. After performing the exchange, the new class's attributes are applied to the object using the specified object attributes mask. Note that object can be any one of type class, drawable or list, but object cannot be of type template.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid object id was used.

SEE ALSO

MAddObject(3C), MDestroyList(3C), MDestroyObject(3C), MRemoveObject(3C)

MFlush

FUNCTION

Flush the output buffer.

SYNTAX

C Interface

```
void MFlush( channel )
        Channel channel;
```

ARGUMENTS

channel	The connection to Cartographer; returned from MOpenChannel.

DESCRIPTION

MFlush flushes the output buffer, causing all buffered requests to be sent to Cartographer.

ERRORS

BadChannel

An invalid channel id was used.

SEE ALSO

MOpenChannel(3C), MCloseChannel(3C) MSync(3C)

MFlushAllEvents

FUNCTION

Flush the event queues for all channels.

SYNTAX

C Interface

```
void MFlushAllEvents()
```

DESCRIPTION

MFlushAllEvents removes all of the events from the input queue. This includes all events received from the Chart Manager, as well as all events placed on the queue by MPut-BackEvent.

MFlushAllEvents differs from MFlush because MFlushAllEvents flushes the events on all channels which this application is connected to, whereas MFlush flushes the requests to a specific Chart Manager.

SEE ALSO

MFlush(3C), MNextEvent(3C), MPending(3C)

MGetProjectionData

FUNCTION

Retrieve the projection data structure in use for the window coordinate system.

SYNTAX

C Interface

```
MapStatus MGetProjectionData( channel, window, pd) Channel  channel;
        WindowId window;
        ProjectionData     *pd;
```

ARGUMENTS

channel	The connection to the Chart Manager; returned from MOpenChannel.
window	The window Id of the window to query.
pd	The projection data structure currently in use for the window.

DESCRIPTION

The MGetProjectionData routine retrieves the current projection data structure for the given window. The routine returns NoError when a valid projection data structure is retrieved and filled in. Normal cases of coordinate conversion should use the calls MPositionToPixels and MPixelsToPo- sition.

RETURN

The function returns a status value indicating whether or not the data structure was retrievable. A status of NoError indicates that the retrieval was successful.

ERRORS

BadChannel

An invalid channel or window was specified.

BadValueError

A convert data structure could not be retrieved because the pointer passed is invalid.

SEE ALSO

MPixelsToPosition(3C), MPositionToPixels(3C)

MGetSearchPath

FUNCTION

Get the current map search path.

SYNTAX

C Interface

```
char **MGetSearchPath( channel, npaths )
Channel  channel;
int *npaths;
```

ARGUMENTS

channel — Specifies the connection to Chart returned by MOpenChannel.

npaths — Assigned the number of map search paths in the list.

DESCRIPTION

The MGetSearchPath returns the current search path the Chart Server is using to get its list of maps for display. The list is returned, along with the number of paths in the list. The space for the path list, and each individual path were allocated using malloc(), and should be freed using free().

ERRORS

BadChannel

An invalid channel id was used.

BadVolume

The specified volume is invalid.

SEE ALSO

MReloadSearchPath(3Map), MAddVolume(3Map), MRemoveVolume(3Map),

MGetServiceContext

FUNCTION

Get copy of service context used by M library.

SYNTAX

ServiceContext MGetServiceContext()

DESCRIPTION

The MGetServiceContext routine returns a reference to the ServiceContext used by the M library. If the library has not yet been initialized, initialization takes place prior to returning to the caller. The ServiceContext can be used directly in any library calls requiring a ServiceContext parameter.

MGetXWindow

FUNCTION

Return the X Window ID of a geographic display window.

SYNTAX

C Interface

```
int MGetXWindow( channel, window )
        Channel  channel;
        WindowId window;
```

ARGUMENTS

channel	Specifies the connection to Chart; returned from MOpenChannel.
window	Specifies the window whose ID is to be returned.

DESCRIPTION

The MGetXWindow function returns the X Window ID of the specified geographic display window. The Chart Manager Window ID and X Server Window ID are unique and different numbers. The Chart Manager Window ID is used to reference a particular window created in Chart. The X Window ID can be used to resize, move, reparent, etc., the X Window containing the geographic display. This function is usually only necessary if a window is to be reparented.

RETURNS

An X Windows Window identifier for the map widget.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

An invalid window id was used.

SEE ALSO

MCreateWindow(3Map), XCreateWindow(3X11)

MKillServer

FUNCTION

Shut down Chart.

SYNTAX

C Interface

```
void MKillServer( channel, kill code )
        Channel   channel;
        int  kill code;
```

ARGUMENTS

channel — The connection to Chart; returned from MOpenChannel.

kill code — A security code that must match the code within Chart.

DESCRIPTION

MKillServer causes Chart to shut down. All connections will be closed, and all windows will be destroyed. The Chart Manager will not quit unless the security code contained in kill code matches the code within Chart.

ERRORS

BadChannel

The channel id was invalid.

SEE ALSO

MOpenChannel(3C), MCloseChannel(3C)

MListFeatures

FUNCTION

List the features available on the Chart Manager.

SYNTAX

C Interface

FeatureListAttributes *MListFeatures(channel, feature type , sub type, nfeatures)

Channel channel; FeatureType feature type; FeatureSubType sub type;
int *nfeatures; /* RETURN */

ARGUMENTS

channel

The connection to the Chart Manager; returned from MOpenChannel.

feature type

The feature type filter for the List of Features. The internal constant AnyFeature will list all features regardless of type.

sub type The sub type filter for the List of Features. The internal constant AnyFeature will list all features regardless of their sub types.

nfeatures The number of records returned from the call, and a corresponding number of records is referenced by the FeatureListAttributes pointer.

DESCRIPTION

The MListFeatures function Lists all of the features that are available in the Chart Manager. The Chart Manager filters the feature list based on the parameters that are passed to it. The wild card AnyFeature can be used to match all feature types.

The information returned by MListFeatures can be used by a Client to display one of the features. The boundary points returned represent the boundary points for the feature as a whole. These will be different values in order for the feature to be displayed in a window. Each

FeatureListAttributes record also returns a set of FeatureAttributes These attributes are the default attributes used by the Draw Module responsible for rendering the feature.

If an error occurs during the query NULL is returned and

nfeatures will be 0.

In the C Version, the memory allocated to store the list of features must be freed by the Client program via a call to MFree.

The feature type and sub type fields should either derive their values from the MuReference(3Mu) routines, or else be set to the constant AnyFeature. Note: this constant cannot be statically assigned.

RETURN

C Interface

A pointer to an array of map structures is returned, with the number of elements in the structure returned in nfeatures. On failure NULL is returned.

STRUCTURES

C Interface

```
typedef unsigned int FeatureType;
typedef unsigned int FeatureSubType;

typedef struct {
    FeatureType  feature_type;
    FeatureSubType    sub_type;
} FeatureProduct;

typedef struct {
    short   line_width;
    short   fill_weight;
    short   fill_offset;
    short   priority;
    MapColor     color;
    MapFillType  fill_type;
    MapLineStyle line_style;
    Boolean show_border;
    Boolean show_bgrd;
    FLOAT   upper_width;
                    char    font_name[ FONT_
```

```
                                NAME_LENGTH  ];

} FeatureAttributes;
typedef struct _FeatureListAttributes { FeatureProduct    type;
   MapLabelAttributes     label;
   FeatureAttributes defaults;
   MapLocationAttributes  location;
} FeatureListAttributes;
```

The fields within the FeatureListAttributes structure are described below:

type

A FeatureProduct structure specification which is used to uniquely specify the feature product. This information can be used by itself to request that the feature be drawn (say within a MAddFeature(3Map) or MChangeMap(3Map) call. This structure contains the following elements:

feature_type

This field indicates the type of feature product to be displayed. These values are internally encoded, and can be referenced using the MuReference(3Mu) utilities. Valid values depend upon the Draw Modules currently connected to the Chart Manager.

sub_type

This field indicates the subfeature to be displayed, given a specific feature type. Similar to the feature type field, this field is internally encoded, and can be referenced using the MuReference(3Mu) utilities.

label

A MapLabelAttributes structure which contains information describing the map product and its coverage. The values contained in this structure are free text values, but generally contain the following information:

name

The place name for the feature, such as "California Roads".

label

A free text field which contains any special comments concerning this feature.

filename

The name of the file where the data header is located. This is generally the location of the feature data as well.

location

A MapLocateAttributes structure describing the coverage of the given feature product. The fields are described as follows:

center
The lat/long value of center point of the feature, as it exists in the database.

top_left
The lat/long value of the top left corner of the feature, as it exists in the database.

bottom_right
The lat/long value of the lower right corner of the feature, as it exists in the database.

supported_projections
A list of projections under which the Draw Module drawing this product can render it. The list may include the value AnyProjection.

num_projections
The number of projections that are supported for the rendering of this feature product. This value is the number of valid items in the supported projections field described above.

scale_recommend
The recommended scale of the feature. The units for this are nautical miles per pixel. This allows all features to be referenced the same way. This value can be used to do rough calculations for items on the screen, but should not be counted on to be accurate as all projections are not necessarily linear.

scale_upper
The upper scale for the feature. The units for this are nautical miles per pixel. This is the largest scale supported by the associated Draw Module rendering the product. Requests to draw the feature larger result in a MaxScale error.

scale_lower
The lower scale for the feature. The units for this are nautical miles per pixel. This is the smallest scale at which the feature will be drawn. Requests to draw the feature smaller result in a

MinScale error.

defaults
A default set of rendering attributes which can be used as a basis for modifying the way in which the feature gets rendered. For a detailed description of each item in the FeatureAttributes structure, refer to the MFeatAtts(3Map) man page.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

The window id used was invalid.

OutOfMemory

Unable to allocate the memory to store the data.

SEE ALSO

MChangeMap(3Map), MListMaps(3Map), MQueryFeatures(3Map)

MListMaps

FUNCTION

List the maps available in the Chart Manager.

SYNTAX

C Interface

```
MapListAttributes *MListMaps( channel, ul, lr, map type , sub type, nmaps )
        Channel  channel;
        MapPoint *ul;
        MapPoint *lr;
        MapType  map type;

        MapSubType    map subtype;
        int *nmaps;       /*  RETURN  */
```

ARGUMENTS

channel The connection to the Chart Manager; returned from MOpenChannel.

ul, lr

The location filter for the List of maps. Only those maps that lie within this boundary are listed. The ul is the upper left point of the bounding box, and the lr is lower right point of the bounding box. To list all maps regardless of the area coverted, set either of the ul or lr fields to the value NULL, or set the elevation field of either point to -1.0.

map type The map type filter for the List of Maps. The internal constant AnyMap will list all maps regardless of type.

sub type The sub type filter for the List of Maps. The internal constant AnyMap will list all maps regardless of their sub types.

nmaps The number of maps contained in the MapList.

DESCRIPTION

The MListMaps function Lists all of the Maps that are available in the Chart Manager. The Chart Manager filters the map list based on the parameters that are passed to it. The wild card AnyMap can be used to match all map types. Also,

a NULL pointer, or an elevation value of -1.0 for the center point can be used as a wild card for center point. This will cause the Chart Manager to match all maps regardless of the area that they cover.

The information returned by MListMaps can be used by a Client to display one of the maps. The boundary points returned represent the boundary points for the map as a whole. These will be different values in order for the map to be displayed in a window. It is best to use the
scale recommend value if this information is to be used in a MChangeMap call. This eliminates the need for the Client program to calculate the boundary points, and worry about correct map aspect ratios.

If an error occurs during the query NULL is returned.

In the C Version, the memory allocated to store the List of Maps must be freed by the Client program via a call to MFree.

RETURN

C Interface

A pointer to an array of map structures is returned, with the number of elements in the structure returned in nmaps. On failure NULL is returned.

STRUCTURES

C Interface

```
typedef unsigned int MapType;

typedef unsigned int MapSubType;

typedef struct {

    MapType map_type;
    MapSubType   sub_type;
} MapProduct;

typedef struct _MapLabelAttributes {

    char   name[
```

NAME_LENGTH + 1];

```
    char   label[ LABEL_LENGTH + 1 ];

    char   filename[ FILE
```

NAME_LENGTH + 1];

```
        } MapLabelAttributes, *MapLabelAttributesList;
```

```
typedef struct _MapLocationAttributes {
    MapPoint    center;
    MapPoint    top_left;
    MapPoint    bottom_right;
    FLOAT       scale_upper;
    FLOAT       scale_lower;
    FLOAT       scale_recommend;
    ProjectionType    supported_projections[NUM_PROJECTIONS]; int
    num_projections;
} MapLocationAttributes, *MapLocationAttributesList;

typedef struct _MapListAttributes {
    MapProduct          type;
    MapLabelAttributes     label;
    MapLocationAttributes  location;
            } MapListAttributes, *MapListAttributesList;
```

The fields within the MapListAttributes structure are described below:

type

A MapProduct structure specification which is used to uniquely specify the map product. In conjunction with the MapLocateAttributes structure, draw requests are formed based upon this information. This structure contains the following elements:

map_type

This field indicates the type of map product to be displayed. These values are internally encoded, and can be referenced using the MuReference(3Mu) utilities. Valid values depend upon the Draw Modules currently connected to the Chart Manager.

sub_type

This field indicates the subtype of product to be displayed, given a specific map type. Similar to the map type field, this field is internally encoded, and can be referenced using the MuReference(3Mu) utilities.

label

A MapLabelAttributes structure which contains information describing the map product and its coverage. The values contained in this structure are free text values, but generally contain the following information:

name

The place name for the map, such as "Camp Pendleton, California".

label
A free text field which contains any special comments concerning this map.

filename
The name of the file where the data header is

located. This is generally the location of the map data as well.

location
A MapLocateAttributes structure describing the coverage of the given map product. The fields are described as follows:

center
The lat/long value of center point of the map, as it exists in the database.

top_left
The lat/long value of the top left corner of the map, as it exists in the database.

bottom_right
The lat/long value of the lower right corner of the map, as it exists in the database.

supported_projections
A list of projections under which the Draw Module drawing this product can render it. The list may include the value AnyProjection.

num_projections
The number of projections that are supported for the rendering of this map product. This value is the number of valid items in the supported projections field described above.

scale_recommend
The recommended scale of the map. The units for this are nautical miles per pixel. This allows all maps to be referenced the same way. This value can be used to do rough calculations for items on the screen, but should not be counted on to be accurate as all projections are not necessarily linear.

scale_upper
The upper scale for the map. The units for this are nautical miles per pixel. This is the largest scale supported by the associated map generator product. Requests to draw the map larger are not complied with.

scale_lower
The lower scale for the map. The units for this are nautical miles per pixel.

This is the smallest scale at which the map will be drawn. Requests to draw the map smaller are not complied

with.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

The window id used was invalid.

OutOfMemory

Unable to allocate the memory to store the data.

SEE ALSO

MChangeMap(3Map), MQueryMap(3Map)

MListObjects

FUNCTION

Object search utility.

SYNTAX

C Interface

```
ObjectListAttributes *MListObjects( channel, atts , mask, nobjects )
        Channel  channel;
                ObjectListSearchAttributes   *atts;
        MapValueMask  mask;
        int *nobjects;       /*  RETURN  */
```

ARGUMENTS

channel — The connection to the Chart Manager; returned from MOpenChannel.

search atts — The criterea used in the search of the Chart Manager object database.

mask — A value mask used to specify those fields in the search criterea which are of interest to the Chart Client.

nobjects — The number of records returned from the call, and a corresponding number of records is referenced by the ObjectListAttributes pointer. In the C Version this pointer is returned as a value by the function, the Ada Version returns this pointer as one of the parameters.

DESCRIPTION

The MListObjects function lists all of the objects that are available in the Chart Manager. The Chart Manager filters the object list based on the search criteria that is passed to it.

The information returned by MListObjects can be used by a Client to query or modify one of the objects. If an error occurs during the query NULL is returned.

In the C Version, the memory allocated to store the list of objects must be freed by the Client program via a call to MFree.

RETURN

A pointer to an array of ObjectListAttributes structures is returned, with the number of elements in the structure returned in nobjects. On failure NULL is returned.

STRUCTURES

C Interface

```
typedef int ObjectType;
typedef int ObjectSubType;
typedef int  ObjectId;
typedef struct {
    ObjectType          type;
    ObjectSubType       subtype;
    ObjectId            class;
    ObjectId            list;

    MapPoint            region[2];
    WindowId            window;
    Boolean             pickability;
    Boolean             visibility;
    } ObjectListSearchAttributes;

typedef struct {

    ObjectId id;
    char     *data;
    } ObjectListAttributes;
```

ERRORS

BadChannel

An invalid channel id was used.

BadValueError

Invalid search criterea specified.

OutOfMemory

Unable to allocate the memory to store the data.

SEE ALSO

MFree(3Map), MQueryObj(3Map)

MMainLoop

FUNCTION

Process input on sockets.

SYNTAX

void MMainLoop( )

DESCRIPTION

The MMainLoop blocks input until it receives a set of descriptors, and then calls the appropriate routines.

MMapWindow

FUNCTION

Map a window to the screen.

SYNTAX

C Interface

```
void MMapWindow( channel, window )
        Channel  channel;
        WindowId window;
```

ARGUMENTS

channel	Specifies the connection to Cartographer; returned from MOpenChannel.
window	Specifies the window id.

DESCRIPTION

The MMapWindow maps a window in Cartographer to the screen. Once created, a window will not be visible until it is mapped to the screen.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

An invalid window id was used.

SEE ALSO

MCreateMapWindow(3C), MUnMapWindow(3C), MDestroyWindow(3C)

MMemory

FUNCTION

Chart Manager memory allocation utilities.

SYNTAX

C Interface

```
char *MAlloc( size)
        unsigned int    size;
char *MReAlloc( ptr, size)
        char  *ptr;
        unsigned int    size;

void MFree( ptrtoptr)
        char  **ptrtoptr;
```

ARGUMENTS

size — The size (in bytes) of the space to be allocated (or reallocated).

ptr — A pointer to the space which has already been allocated.

ptrtoptr — A pointer to the pointer to the space which has been allocated. This is used by MFree() in lieu of simply passing ptr so that the address pointer then can be set to NULL. This prevents multiple calls to MFree() from freeing the same address twice, which can produce unpredictable (and sometimes hard to find) results.

DESCRIPTION

The MMemory calls provide a mechanism for allocating and freeing memory. They are used only with the C interface.

MReAlloc() reallocates the memory block pointed to by ptr to be size bytes. If size is greater than the block's previous size, then the contents pointed to by ptr are copied to the new block. If size is smaller than the block's previous size, then size bytes of the contents pointed to by ptr are copied to the new block. In either case, the space pointed to by ptr is then freed. MReAlloc() returns NULL in cases where the allocation fails.

MFree() frees up a block of memory which was allocated by MAlloc() or MReAlloc(). Note that the parameter passed to MFree() is the address of the block pointer, rather than the address of the block itself. Once the block is freed the block pointer address is set to NULL. Then if MFree() is called again with the same block pointer address, it will prevent freeing the same block twice, which can lead to unpredictable results.

MDebug discusses some debugging features which are available with the MMemory routines.

RETURN

Both MAlloc and MReAlloc return a pointer to the newly allocated space. They return NULL if the call fails for some reason.

SEE ALSO

MDebug(3Map)

MModifyFeature

FUNCTION

Modify feature attributes on a map.

SYNTAX

C Interface

```
void MModifyFeature( channel, window, feature, mode ) Channel   channel;
        WindowId  window;
                MapFeatureAttributes     *feature;
        ModificationMode    mode;
```

ARGUMENTS

channel	The connection to the Chart Manager; returned from MOpenChannel.
window	The window on which the feature is to be displayed.
feature	A map feature whose attributes are to be modified. See MFeatAtts(3Map) for information on this structure's contents.
mode	The mode to use when modifying the feature attributes. The mode can be one of four values: ModifyAnd, ModifyOr, ModifyXor, and ModifySet. See MModifyFeatures(3Map) for more information on this field.

DESCRIPTION

MModifyFeature modifies the drawing attributes of the specified feature onto the currently drawn map. The effects of this call are not seen until the map gets redrawn. Note that this call does not cause the map to redraw! Clients should use the MUpdateFeatures call to see the changes take effect. Not all feature attributes have an effect on the feature as it gets drawn. The effect of the feature attributes depends on the Draw Module responsible for drawing the particular feature.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

An invalid window id was used.

BadValueError
An invalid or non-existent feature value was specified. Also occurs if a bad ModificationType is specified.

SEE ALSO
MAddFeature(3Map), MAddFeatures(3Map), MChangeMap(3Map), MFeatAtts(3Map), MFeatMask(3Map), MModifyFeatures(3Map), MRemoveFeature(3Map), MRemoveFeatures(3Map), MuReference(3Mu)

MModifyFeatures

FUNCTION

Modify a list of feature attributes on a map.

SYNTAX

C Interface

```
void MModifyFeatures( channel, window, feature,
    nfeatures, mode)
        Channel   channel;
        WindowId  window;
                MapFeatureAttributes    *feature;
        int  nfeatures;
        ModificationMode    mode;
```

ARGUMENTS

channel	The connection to the Chart Manager; returned from MOpenChannel.
window	The window on which the features are to be displayed.
feature	A list of map features whose attributes are to be modified on the map. See MFeatAtts(3Map) for information on this structure's contents.
nfeatures	The number of map features being modified.
mode	The mode to use when modifying the feature attributes. The mode can be one of four values: ModifyAnd, ModifyOr, ModifyXor, and ModifySet. See STRUCTURES section below for more detail on each value.

DESCRIPTION

MModifyFeatures modifies the drawing attributes of the specified features onto the currently drawn map. The effects of this command are not seen until the map is redrawn. The client should call MUpdateFeatures() to see the effects of the modified feature attributes. Not all feature attributes have an effect on the feature as it gets drawn. The effect of the feature attributes depends on the Draw Module responsible for drawing the particular feature.

STRUCTURES

C Interface

typedef int ModificationMode;

The ModificationMode is used to specify how a Chart Client modifies FeatureAttributes for a feature already present on the display list. Four modification types are permitted: ModifyAnd, ModifyOr, ModifyXor, and ModifySet. Each of these affects the modification differently, depending on what FeatureAttributes have been previously set using one of the M library calls which affect a feature's display attributes.

The display list of features for a given map window is affected by the following calls: MModifyFeature(), MModifyFeatures(), MAddFeature(), MAddFeatures(), MRemoveFeature(), MRemoveFeatures(), and MChangeMap(). When a feature is added to a geographic display using MAddFeature(s)() or MChangeMap(),it features even if the feature is not currently viewable. Chart Clients can modify the attributes used to render a viewable feature using the MModifyFeature() and MMo difyFeatures() calls.

A bitmask of FeatureAttributes is saved, along with the FeatureAttributes themselves for each feature in the display list. This bitmask is simply a representation of those features which have been supplied by a Chart Client, and gets modified using the MModifyFeature(s)() calls. The ModificationMode expresses how the attributes supplied in the current MModifyFeature(s)() call affects the feature attributes on the display list.

For example, when set to ModifyAnd, only those feature attributes common to the FeatureAttributes structure in the supplied command, and to the FeatureAttributes structure in the display list, are updated.

The supported values for modification mode are the following:

ModifyAnd

ModifyAnd indicates that only those fields in the supplied FeatureAttri butes which are in common with the current set of Client FeatureAttributes are to be modified. The feature modification bitmask is not updated in this case because no new fields are modified.

ModifyOr

ModifyOr indicates that those fields supplied in the FeatureAttributesfP structure are to be

unconditionally modified. The feature attributes modification mask is updated to indicate that these fields have been set by a Chart Client.

ModifyXor

ModifyXor indicates that only those fields which haven't been previously m odified, and which are being modified now, are to be modified. The feature attributes modification mask is updated to indicat that these fields have been set by a Chart Client.

ModifySet

ModifySet is used to unconditionally set the FeatureAttributes to the supplied values. The feature modification mask is set to the mask value supplied with the FeatureAttributes in this call.

Note that fields within the FeatureAttributes structure which the Chart Client(s) fail to specify take on their default values, which are usually supplied by the responsible (rendering) Draw Module. For the most part, these values are the best combination of attributes for rendering the particular feature. User customization is available, using these calls, however.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

An invalid window id was used.

BadValueError

An invalid or non-existent feature value was specified. Also occurs if a bad ModificationMode is specified.

SEE ALSO

MAddFeature(3Map), MAddFeatures(3Map), MChangeMap(3Map), MFeatAtts(3Map), MFeatMask(3Map), MModifyFeature(3Map), MRemoveFeature(3Map), MRemoveFeatures(3Map), MuReference(3Mu)

MModifyObject

FUNCTION

Modify an already existing object through animation.

SYNTAX

C Interface

```
void MModifyObject( channel, object, animation type, flags, modifier )
    Channel   channel;
    ObjectId  object;
    int  animation type;
    int  flags;
    int  modifier;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel().

object — Specifies the ID of the object to be modified. This object must be a Drawable.

animation type

Specifies the type of modification desired on the object. Legal values for this field depends on the object being modified. Possible values include:

DragObject
: Move the entire object.

EndPoint
: Moves an object's end point.

InsertLeg
: Inserts a new line segment into the object.

MajorAxis
: Modifies an object's major axis of rotation.

MinorAxis
: Modifies an object's minor axis of rotation.

MoveVertex
: Moves one of the object's vertices.

Size

Modifies an object's dimensions.

flags Object-specific modifier.

modifier Specifies a particular point, vertex, or leg of an object for certain modification types.

DESCRIPTION

The MModifyObject() function will modify an object of the specified type using animation. Valid objects which can be modified via animation include:

Arc

This function requires animation type to specify one of three possible modification types: Size, EndPoint, or DragObject. In the case where animation type is set to EndPoint, then modifier is set to one of the following :

0 - point at start of arc from true north

1 - point at extent of arc from true north

Box

This function requires animation type to specify one of two possible modification types: Size or DragObject. In the case where animation type is set to Size, the modifier value determines which corner of the rectangle is selected for modification. In the case of a bearing value of 0 degrees, vertex 0 is the upper left corner, vertex 1 is the upper right corner, vertex 2 is the lower right corner, and vertex 3 is the lower left corner. In all cases the animation will insure that the rectangular shape is retained.

Circle

This function requires animation type to specify one of two possible modification types: Size or DragObject.

Ellipse

This function requires animation type to specify one of three possible modification types: MajorAxis, MinorAxis, or DragObject.

Line

This function requires animation type to specify End- Point, and for modifier to specify 0 or 1 to indicate which endpoint should be modified.

Polygon

This function requires animation type to specify either InsertLeg or MoveVertex. In either case, modifier is used for specifying which leg or vertex is to be

modified.

Polyline

This function requires animation type to specify either InsertLeg or MoveVertex. In either case, modifier is used for specifying which leg or vertex is to be modified.

Rectangle

This function requires animation type to specify End- Point, and for modifier to specify 0, 1, 2 or 3 to indicate which endpoint on the rectangle should be modified. Vertex 0 is considered to be the first point specified in the MDrawRectange() call, or the first point clicked during a Rectangle object create.

Sector

This function requires animation type to specify one of two possible modification types: EndPoint or DragOb- ject. In the case where animation type is set to End- Point, then modifier is set to one of the following :

0 - point on inner radius at start of arc

1 - point on outer radius at start of arc 2 - point on outer radius at end of arc 3 - point on inner radius at end of arc.

Text

This function requires animation type to specify Dra- gObject, since the position of the text is the only thing that can be modified.

MModifyObject() makes use of the animate keys which are defined by the last call to MSetAnimateKeys(). If the escape key is pressed anytime during the animate process, then the animation is aborted, and no modifications are made. New animation points are selected using either the first button on the input device, or by positioning the cursor at the desired point and hitting the select key. The finish key does the same thing in this case.

Upon successful modification of an object, an ObjectChangedEvent is sent to the application which requested the modification.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid template id was used.

BadOwner

An animation is already under way.

BadValueError

An invalid object type was specified.

ObjectNotVisible

The object to be modified is not visible on the screen. All modified objects must be visible.

SEE ALSO

MAbortAnimation(3C), MCreateObject(3C), MEvents(3C), MObjAtts(3C), MSetAnimateKeys(3C)

MMoveObject

FUNCTION

Move an object in a window.

SYNTAX

C Interface

```
void MMoveObject( channel, object, move info );
Channel   channel;
ObjectId  object;
        MapSetMoveAttributes     *move info;
```

ARGUMENTS

channel Specifies the connection to Cartographer; returned from MOpenChannel().

object Object Id of the object to be moved.

move info A description of where to move the object.

DESCRIPTION

The MMoveObject() function moves an object on the Map. The move info structure describes how to move the object. The new position of the object can be determined in several ways. The first element (move type) of the move info structure determines how the object will be moved. Shown below is a list of possible values and their meaning.

MoveAbsolute

the object is moved to the point specified in location.

MoveRelative

the point specified in location is added to the current location of the object.

MoveBearing

The new location for the object is calculated using the bearing and distance elements of the structure.

STRUCTURES

C Interface

```
typedef struct {
   int     move_type;
   MapPoint     position;
} SetMovePosition;
```

```
typedef struct {
    int         move_type;
    FLOAT       distance;
    FLOAT       bearing;
} SetMoveBearing;

typedef union {
    int         move_type;
    SetMovePosition   absolute;
    SetMovePosition   relative;
    SetMoveBearing    bearing;
} MapSetMoveAttributes;
```

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid object id was used.

BadValueError

An invalid move type was specified.

SEE ALSO

MCreateObject(3C), MDestroyObject(3C), MDestroyList(3C), MModifyObject(3C)

MNextEvent

FUNCTION

Get the next event from the event queue.

SYNTAX

C Interface

```
void MNextEvent( report )
        MapEvent   *report;
```

ARGUMENTS

report The next event in the queue.

DESCRIPTION

The MNextEvent function copies the first event from the event queue into the specified MapEvent structure and then removes it from the queue. If the event queue is empty, MNextEvent blocks until an event is received. MNextEvent is set up to receive events from more than one MapServer if the application happens to be connected to more than one.

SEE ALSO

MOpenChannel(3C), MPending(3C), MPutBackEvent(3C)

MNoOp

FUNCTION

No action is perfomed by Cartographer.

SYNTAX

C Interface

```
void MNoOp( channel )
        Channel channel;
```

ARGUMENTS

channel	Specifies the connection to Cartographer; returned from MOpenChannel.

DESCRIPTION

The MNoOp function causes no actions in Cartographer and is provided mainly to exercise the communication channel.

ERRORS

BadChannel

An invalid channel id was used.

SEE ALSO

MOpenChannel(3C), MCloseChannel(3C)

MOpenChannel

FUNCTION

Open a channel for communication to Cartographer.

SYNTAX

C Interface

```
Channel MOpenChannel( node )
        char  *node;
```

ARGUMENTS

node — The name of the machine to connect to.

DESCRIPTION

The MOpenChannel function opens a channel for communication between the Cartographer Client and Cartographer. A Channel ID is returned and is used to reference the connection with Cartographer. Every call made to the library requires a channel identifier as a parameter. If the library is unable to connect to Cartographer, then InvalidChannel is returned.

If the node parameter is NULL, MOpenChannel will attempt to connect to Cartographer on the same machine as the Client.

RETURN VALUE

Upon successful connection, the channel ID of the connection is returned; otherwise InvalidChannel is returned.

ERRORS

OutofMemory

Unable to allocate space for this channel. No memory left.

AlreadyConnected

The Cartographer Client already has an open connection to this Cartographer Manager. No more than one connection per Cartographer Manager is allowed for each Cartographer Client.

SEE ALSO

MCloseChannel(3C)

MPending

FUNCTION

Return the number of pending Map events.

SYNTAX

C Interface

int MPending( )

DESCRIPTION

The MPending function returns the number of input events that have been received from Cartographer, but not yet removed from the event queue.

RETURN

The function returns the number of events still on the event queue. A value of 0 is returned if no events are presently on the queue.

SEE ALSO

MPutBackEvent(3C)

MPixelsToPosition

FUNCTION

Convert a window point to geodetic coordinates (lat/long).

SYNTAX

C Interface

```
Boolean MPixelsToPosition ( channel, window, i1, p1 ) Channel  channel;
        WindowId window;
        IntMapPoint   *i1;
              MapPoint *p1;       /*  RETURN  */
```

channel	The connection to Cartographer; returned from MOpenChannel.
window	The window to be queried.
i1	The point to be converted, in window coordinates.
p1	The geodetic coordinates value of the point.

DESCRIPTION

The MPixelsToPosition function converts a pixel point on the geographic display to its geodetic coordinates. If the specified pixel position does not correspond to a point on the viewable geographic display surface, then False is returned; otherwise True is returned. The current projection and scale are taken into account when converting the pixel location to a geodetic coordinate. The pixel locations are mapped to the current geographic display window space, where (0,0) represents the upper left hand corner of the window, and (width, height) represents the lower right hand corner. Negative pixel values represent points above and to the left of the upper left hand corner, and may be valid provided that the viewable geographic display space extends beyond the corners of the window. Likewise, values greater than width or height may occur if the viewable map space extends below or to the right of the window.

RETURN

The value True is returned when a valid conversion takes place, and the pixel location lies on top of the viewable geographic display space. The value False is returned when the pixel location lies outside the viewable geographic display space, or is otherwise not convertible.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId
The window id used was invalid.

BadValueError
The value (in window coordinates) was invalid.

SEE ALSO
MPositionToPixels(3C),

MPositionToPixels

FUNCTION

Convert a geodedic coordinate (lat/long) point to pixel coordinates.

SYNTAX

C Interface

```
Boolean MPositionToPixels( channel, window, p1, i1 ) Channel  channel;
        WindowId window;
        MapPoint *p1;
        IntMapPoint   *i1;      /*   RETURN  */
```

ARGUMENTS

channel	The connection to Cartographer; returned from MOpenChannel.
window	The window Id of the window to query.
p1	The geodedic coordinate point to be converted.
i1	The location of the point in window coordinates.

DESCRIPTION

The MPositionToPixels function converts a geodedic coordinate point to window coordinates. The point is not guaranteed to be in the window. If the point lies within the bounds of the displayable geographic display surface,
True is returned; otherwise, False is returned. The conversion takes into account the current projection and scale of the geographic display. The output pixel values are mapped to the given window so that the pixel location (0,0) corresponds to the upper left corner of the window, and the pixel location (width, height) corresponds to the lower right corner of the window. Negative values for either coordinate may occur if the viewable geographic display space extends above or to the left of the window. Likewise, values greater than width or height may occur if the viewable geographic display space extends below or the the right of the window.

RETURN

The function returns a Boolean value indicating whether or

not the point lies within the viewable space. For example, if the function returns False, then the point is not viewable on the current geographic display. A value of True indicates

that the point does lie on the viewable space.

STRUCTURES

C Interface

```
typedef struct _IntMapPoint {
    int     x;
    int     y;
    int     z;
} IntMapPoint, *IntMapPointPtr;
```

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

The window id used was invalid.

BadValueError

The specified location was invalid.

SEE ALSO

MPixelsToPosition(3C),

MPutBackEvent

FUNCTION

Push an event back on the input queue.

SYNTAX

C Interface

```
void MPutBackEvent( event )
        MapEvent *event;
```

ARGUMENT

event Specifies a pointer to the event to be requeued.

DESCRIPTION

MPutBackEvent pushes an event back onto the head of the input queue (so that it would become the next one returned by the MNextEvent call). This can be useful if you read an event and then decide that you would rather deal with it later. There is no limit to the number of times in succession that you can call MPutBackEvent.

SEE ALSO

MNextEvent(3C), MPending(3C), MSendEvent(3C)

MQueryChannel

FUNCTION

Get information about a channel.

SYNTAX

C Interface

```
ChannelInfo *MQueryChannel( channel )
        Channel  channel;
```

ARGUMENTS

channel	Specifies the connection to Cartographer; returned by MOpenChannel.

DESCRIPTION

The MQueryChannel provides a Cartographer Client with information on the specified communications Channel.

RETURN

MQueryChannel() returns a pointer to a ChannelInfo structure which is allocated in the C version. This structure should be freed by calling free when done using the structure.

STRUCTURES

C Interface

```
typedef struct _ChannelInfo {
    int   num_connections;
    int   num_windows;
    char  node[NODE_
```

NAME_LENGTH];
} ChannelInfo;

The contents of the ChannelInfo structure are as follows:

num_connections
The current number of connections to Cartographer.

num_windows
The total number of defined windows for this Cartographer Manager.

node The host name of the machine that Cartographer is running on.

ERRORS

BadChannel
The channel id is invalid

OutofMemory
Unable to allocate space for ChannelInfo structure.

SEE ALSO
MOpenChannel(3C),

MQueryFeatures

FUNCTION

Retrieve display list of features for given geographic display.

SYNTAX

C Interface

MapQueryFeatures *MQueryFeatures(channel, window, feature type, feature subtype)

Channel channel;
WindowId window;
FeatureType feature type; FeatureSubType feature subtype;

channel : in MTypes.Channel; window : in MTypes.WindowId;
feature type : in MTypes.FeatureType;
feature subtype : in MTypes.FeatureSubType) return MTypes.MapQueryFeatures;

ARGUMENTS

channel — Specifies the connection to the Chart Manager; returned from MOpenChannel.

window — Specifies the window to be queried.

feature type — Specifies the type of feature product to be queried. The value AnyFeature is supported, and results in querying for all features regardless of type.

feature subtype — Specifies the subtype of feature product to be queried. The value AnyFeature is supported, and results in querying for all features regardless of subtype.

DESCRIPTION

The MQueryFeatures routine is used by Chart Clients to get a current list of displayed features for a given geographic display (map window). This display list may include features which are not currently visible because certain feature rendering attributes preclude them from being visible. Refer to the MFeatAtts(3Map) manual page.

RETURN

The MQueryFeatures function returns a description of the features which are currently displayed in the indicated window. If an error occurs NULL is returned. MapQueryFeatures is allocated within this routine. It is the responsibility

of the calling process to free the memory using MFree (C

only).

STRUCTURES

C Interface

```
typedef struct {
        int  num_features;
                MapFeatureAttributes     *features;
} MapQueryFeatures;
```

The fields for the MapQueryAttributes structure are described below:

features

A pointer to a list of currently displayed features, and the attributes under which they have been rendered. A full description of each field in this structure may be found under MFeatAtts(3Map). There are no built-in limits as to the number of features which may be returned by an MQueryFeatures() call. In fact, under certain circumstances the number can be rather large (such as the display of countries at World View).

num_features

The number of features specified in the features list.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

The window id used was invalid.

OutOfMemory

Unable to allocate the memory to store the data.

SEE ALSO

MChangeMap(3Map), MEvents(3C), MFeatAtts(3Map), MListFeatures(3Map), MQueryMap(3Map), MuReference(3Mu)

MQueryMap

FUNCTION

Get current geographic display attributes for a given map window.

SYNTAX

C Interface

```
MapQueryAttributes *MQueryMap( channel, window, mode ) Channel  channel;
        WindowId window;
        MapBoundaryMode    mode;
```

ARGUMENTS

channel	Specifies the connection to the Chart Manager; returned from MOpenChannel.
window	Specifies the window to be queried.
mode	Specifies the boundary mode type to be returned. Valid values are: UseScale, UseWidth, and UseBoundary. See MBoundary for more details on the supported modes.

DESCRIPTION

The MQueryMap function is used to obtain the current state of the geographic display. This includes: the geographic display's coverage, a list of displayed map products, applicable color models in use, and the displayed map projection. The feature display list is NOT included here. Chart Clients needing this information should use the MQueryFeatures call.

RETURN

The MQueryMap function returns a pointer to a MapQueryAttributes structure. If an error occurs NULL is returned. The memory to hold the MapQueryAttributes is allocated within this routine. It is the responsibility of the calling process to free the memory using MFree (C only).

STRUCTURES

C Interface

```
typedef struct {
    int   num_products;
    MapProductAttributes products[MAX_MAPS]; ProjectionType  projection;

            MapBoundaryAttributes     boundary;
```

```
        int   num_colors;
        MapColorAttributes   color[MAX_COLOR_MODELS]; FLOAT
        zoom_factor;
} MapQueryAttributes;
```

The fields for the MapQueryAttributes structure are described below:

products
: The specifications for the map products in the geographic display. This structure is described in detail under MProdAtts(3Map). Even though they are they same structure, the product specification returned by MQuer- yMap tends to be more specific than that which is provided by MChangeMap.

num_products
: The number of products specified in the products list. This value will not exceed the Chart Manager constant MAX MAPS.

projection
: A projection being displayed. The MProjection(3Map) man page discusses display projections in detail.

color
: A list of color specifications for the map display. The MColor(3Map) man page discusses color models in detail. This specification is used only if the CMSetColor bit is set in value mask.

num_colors
: The number of color specifications in the color list.

  This value cannot exceed the Chart Manager constant MAX COLOR MODELS.

boundary
: The boundaries of the map to be displayed. The MBoundary(3Map) man page discusses map display boundaries in detail. This specification is used only if the CMSetBoundary bit is set in value mask.

zoom_factor
: This is the current zoom factor retrieved by the most recent call to MQuickZoom(3C). If this routine has not been used, then this value will always be 1.0. If this routine has been used, then this value will lie between 0.0 and 1.0. In all cases where a quick zoom does occur, the map's boundary attributes are updated in the boundary fields.

ERRORS

BadChannel
An invalid channel id was used.

BadWindowId
The window id used was invalid.

OutOfMemory
Unable to allocate the memory to store the data.

SEE ALSO
MBoundary(3Map), MChangeMap(3Map), MColor(3Map), MEvents(3C), MFeatAtts(3Map), MListMaps(3Map), MProdAtts(3Map), MProjection(3Map), MQueryFeatures(3Map), MQuickZoom(3C), MuReference(3Mu)

MQueryObject

FUNCTION

Get information about an object.

SYNTAX

C Interface

```
MapObjectAttributes *MQueryObject( channel, object ) Channel  channel;
                    ObjectId object;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

object — The object being queried.

DESCRIPTION

MQueryObject is used to query the attributes of the given object.

RETURN

MQueryObject returns a pointer to a MapObjectAttributes structure. Users of the C routine must free this space via a call to free when done. If the call fails, then NULL is returned.

STRUCTURES

C Interface

```
typedef int ObjectType;
typedef int ObjectSubType;
typedef struct {
        ObjectType type;
        ObjectSubType   subtype;
        ObjectAttributes     atts;
        XRectangle bounding_box;
        ObjectData object_data;
        } MapObjectAttributes;
```

The MapObjectAttributes structure contains the following items:

type

The object type. Can be one of List, Class, Template. or Drawable.

subtype

The object subtype. Can be one of Arc, Bitmap, Box, Circle, Ellipse, Line, Polygon, Polyline, Rectangle, Segment, Symbol, or Text.

atts

The ObjectAttributes structure for this object; see MObjAtts(3C) for more detail.

bounding_box

A rectangular bounding box which describes, in pixels, the maximum breadth of the object on the viewing screen. The structure used to describe this is an XRectangle structure, which is described in the MQueryObjectBBox(3C) manual page.

object_data

Specific data used to create the object. The MObject-Data manual page describes the ObjectData structure in detail.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid object id was used.

SEE ALSO

MObjAtts(3C), MObjectData(3C), MQueryObjectBBox(3C)

MQueryObjectBBox

FUNCTION

Get an object's coverage box.

SYNTAX

C Interface

```
XRectangle *MQueryObjectBBox( channel, window, objects nobjects )
        Channel  channel;
        WindowId window;
        ObjectId *objects;
        int nobjects;
```

ARGUMENTS

channel	Specifies the connection to Cartographer; returned from MOpenChannel.
window	Specifies the window on which the object to query is currently displayed.
objects	The objects being queried.
nobjects	The number of objects to query.

DESCRIPTION

MQueryObjectBBox is used to query the bounding box coverage of the given objects. This call is more efficient than the more general purpose MQueryObject() call, when the Cartographer Client is only interested in the area of coverage for a given object.

RETURN

MQueryObjectBBox returns a pointer to an XRectangle structure. Users of the C routine must free this space via a call to free when done. If the call fails, then NULL is returned.

STRUCTURES

C Interface

```
typedef struct {
        short x;
        short y;
        unsigned short  width;

        unsigned short  height;
```

} XRectangle;

The XRectangle structure contains the following items:

x, y
: The object's upper left corner point, in pixels. This point is relative to the upper left corner of the display pixmap, not the window pixmap, which may or may not exceed the bounds of the display window.

width, height
: The object's extent. Both values are in pixels.

ERRORS

BadChannel
: An invalid channel id was used.

BadObjectId
: An invalid object id was used.

NOTES

The returned bounding box has values in pixels, relative to the display pixmap. These are somewhat different than the values returned by MPositionToPixels(3C), as the display pixmap is usually larger than the window size. The scaling between the two pixmaps is the same, however. Hence only the (x,y) upper left coordinate is affected by this.

SEE ALSO

MQueryObject(3C)

MQueryWindow

FUNCTION

Get information about a Map Window.

SYNTAX

C Interface

QueryWindowAttributes *MQueryWindow(channel, window) Channel channel;
WindowId window;

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

window — The window to be queried.

DESCRIPTION

The MQueryWindow returns the attributes of the specified window. Refer to MWindowAtts(3C) for more information on this structure.

RETURNS

The C function returns a NULL pointer in the event that an error occurs. Otherwise a pointer to a WindowAttributes structure is returned. For the C interface, the returned space is allocated using malloc. This space should be freed up using free.

ERRORS

BadChannel
An invalid channel id was used.

BadWindowId
An invalid window id was used.

BadOwner
The window exists, but this process has never connected to it.

OutofMemory
Unable to allocate space for WindowAttributes structure.

SEE ALSO

MCreateMapWindow(3C), MDestroyWindow(3C),

MQuickZoom

FUNCTION

Quickly zoom current view.

SYNTAX

C Interface

```
void MQuickZoom( channel, window, zoom factor )
Channel   channel;
WindowId  window;
FLOAT     zoom factor;
```

ARGUMENTS

channel — The connection to Cartographer; returned from MOpenChannel.

window — The window to which the quick zoom is applied.

zoom factor — A relative scale factor at which the map is to be rendered. Valid range is from 0.1 to 1.0.

DESCRIPTION

MQuickZoom requests that the current view be quickly zoomed. Quick zooming re-renders the current view using pixel replication. The zooming is always performed relative to the scale of the last map drawn. This re-rendering is limited to be 10% of the current map scale (0.1).

Upon successful completion of this call, interested Clients are sent an UpdateCoverageNotify event.

ERRORS

AlreadyDrawingMap

A map draw is already in progress.

BadValueError

A zoom factor outside the range limits was speicified. This might also occur if the new produced scaling factors fail.

SEE ALSO

BUGS

MQuickZoom is not all that fast on some machines.

MRecenterMap

FUNCTION

Recenter a map.

SYNTAX

C Interface

```
void MRecenterMap( channel, window, new center ) Channel  channel;
                     WindowId window;
                     MapPoint *new center;
```

ARGUMENTS

channel	Specifies the connection to Cartographer; returned from MOpenChannel.
window	Specifies the window to be queried.
new center	The new point on which to center the map display.

DESCRIPTION

The MRecenterMap function recenters the current map display on the specified point. All other items remain the same, except for the product list, which may be modified slightly. The products which are used are the current product list, with the subtype set to AnyMap. This allows for other map products in the same class to be seamlessly rendered based on the current location in the world.

ERRORS

SEE ALSO

MReleaseFocus

FUNCTION

Release point select focus.

SYNTAX

C Interface

```
void MReleaseFocus ( channel, window )
        Channel   channel;
        WindowId  window;
```

ARGUMENTS

channel	Specifies the connection to Cartographer; returned by MOpenChannel.
window	The window whose focus is being released.

DESCRIPTION

MReleaseFocus stops PointSelectEvents from being sent to this process. These events are requested through the call MRequestFocus.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

The window id passed is not valid.

BadOwner

The window is valid and exists, but the user has never connected to it.

SEE ALSO

MCreateMapWindow(3C), MOpenChannel(3C), MSendEvent(3C), MSetEventMask(3C), MRequestFocus(3C)

MReleaseWindow

FUNCTION

Release Cartographer Client as a user of a window.

SYNTAX

C Interface

```
void MReleaseWindow( channel, window )
        Channel  channel;
        WindowId window;
```

ARGUMENTS

channel	The connection to Cartographer; returned by MOpen-Channel.
window	The window to be released.

DESCRIPTION

MReleaseWindow removes this channel as a user of the window. The application will no longer be sent events about this window. If this application is the last user of a window, the window will be destroyed.

The window's owner is defined to be the Client which created the window. If the window's owner releases its connection using MReleaseWindow, then the window no longer has an owner. Other users of the window will be able to continue to use the window until the last user releases its connection, at which point the window is destroyed. If the window's owner instead releases its connection using MDestroyWindow, then the window is unconditionally destroyed, and all other window connections are terminated.

ERRORS

BadChannel

The channel id was invalid.

BadWindowId

The specified window id is invalid.

BadOwner

The specified window id is valid, but this Client does not have an established connection to it.

SEE ALSO

MDestroyWindow(3C), MUseNamedWindow(3C), MUseWindow(3C)

MReloadSearchPath

FUNCTION

Cause Chart to reload its lists of volumes to search for maps.

SYNTAX

C Interface

```
void MReloadSearchPath( channel )
        Channel    channel;
```

ARGUMENTS

channel	Specifies the connection to Chart; returned from MOpenChannel.

DESCRIPTION

The MReloadSearchPath causes Chart to reload its search path. It will look at the environment variable MapSear- chPath for its list of directories. If this variable is not present, Chart will load the defaults that were specified at compile time. The map search path can be further modified through the calls MAddVolume and MRemoveVolume.

ENVIRONMENT

MapNoRecursion

When this environment variable is set, only the specified path(s) is/are checked for map files. Otherwise the specified path(s) and all of its/their subdirectories are checked.

MapSearchPath

This environment variable is looked at to obtain a path list for loading maps.

ERRORS

BadChannel

An invalid channel id was used.

BUGS

If a Chart Client calls this routine, existing map color resources are freed up. This can occur even though one or more map windows are present and visible. The colors will be reallocated upon the next map draw command. In the meantime, some map colors could unexpectedly change. This effect will only be seen on PseudoColor and GrayScale machines.

SEE ALSO

MAddVolume(3Map), MRemoveVolume(3Map),

MRemoveFeature

FUNCTION

Remove specified feature from a map.

SYNTAX

C Interface

```
void MRemoveFeature( channel, window, feature )
Channel   channel;
WindowId  window;
FeatureProduct *feature;
```

ARGUMENTS

channel	The connection to the Chart Manager; returned from MOpenChannel.
window	The window where intensity is to be set.
feature	A map feature to be removed from the map. The FeatureProduct structure is described below.

DESCRIPTION

MRemoveFeature removes the specified feature from the currently drawn map. The MListFeatures(3Map) call provides a list of supported features in the Chart Manager. This command and others which convert FeatureType types to strings and vice-versa are documented under MuReference(3Mu).

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

An invalid window id was used.

BadValueError

An invalid or non-existent feature value was specified.

SEE ALSO

MAddFeature(3Map), MAddFeatures(3Map), MChangeMap(3Map), MFeatAtts(3Map), MModifyFeature(3Map), MModifyFeatures(3Map), MRemoveFeatures(3Map), MuReference(3Mu)

MRemoveFeatures

FUNCTION

Remove specified features from a map.

SYNTAX

C Interface

```
void MRemoveFeatures( channel, window, features, nfeatures )
    Channel   channel;
    WindowId  window;
    FeatureProduct *features;
    int  nfeatures;
```

ARGUMENTS

channel The connection to the Chart Manager; returned from MOpenChannel.

window The window where intensity is to be set.

features A list of map features to be removed from the map. nfeatures The size of the feature list specified in feature.

DESCRIPTION

MRemoveFeatures removes the specified features from the currently drawn map. The MListFeatures(3Map) call provides a list of supported features in the Chart Manager. This command and others which convert FeatureProduct types to strings and vice-versa are documented under MuReference(3Mu).

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

An invalid window id was used.

BadValueError

An invalid or non-existent feature value was specified.

SEE ALSO

MAddFeature(3Map), MAddFeatures(3Map), MChangeMap(3Map), MFeatAtts(3Map), MModifyFeature(3Map), MModifyFeatures(3Map), MRemoveFeature(3Map), MuReference(3Mu)

MRemoveInput

FUNCTION

Remove an input source from the Chart Manager.

SYNTAX

```
#include <M/Service.h>

void MRemoveInput( fd, mask )
        int  fd;
        int  mask;
```

ARGUMENTS

fd	Specifies the source file descriptor on a Unix based system..
mask	Specifies the condition mask that tells when the routine should be called. Valid entries for this field are ServiceReadMask, ServiceWriteMask, ServiceExceptMask, or ServiceAllMask.

DESCRIPTION

The MRemoveInput() routine removes a monitor for a data source. The callback for the condition is cleared, even if an input handler has not been previously added.

The condition is specified by a bit field mask. This mask is created by or-ing any of the following: ServiceReadMask, ServiceWriteMask, or ServiceExceptMask. Also, ServiceAllMask can be specified. This is defined to be all three operations.

SEE ALSO

MAddInput(3Map)

MRemoveObject

FUNCTION

Remove an object from a list.

SYNTAX

C Interface

```
void MRemoveObject( channel, list, object )
Channel   channel;
ObjectId  list;
ObjectId  object;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

list — Specifies the list through which to search for the given object.

object — Specifies the object to be removed from the list.

DESCRIPTION

MRemoveObject removes the specified object from it's list. The ObjectAttributes of both the object and the List remain unaffected. Also the object that is removed from the list is not destroyed.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid object id was used.

SEE ALSO

MAddObject(3C), MDestroyObject(3C), MCreateList(3C), MListObjects(3C)

MRemoveProduct

FUNCTION

Remove one specified map product from a geographic display.

SYNTAX

C Interface

```
void MRemoveProduct( channel, window, product )
Channel   channel;
WindowId  window;
ProductId product;
```

ARGUMENTS

channel	The connection to the Chart Manager; returned from MOpenChannel.
window	The window on which the product is currently displayed.
product	A map product to be removed from the geographic display.

DESCRIPTION

MRemoveProduct removes the specified product from the currently drawn geographic display. Products which are drawn on the current display are referenced using ProductIds. These values are returned as part of the product's MapProductAttributes structure.

ERRORS

AlreadyDrawingMap
: A map draw command is already in progress for the specified window.

BadChannel
: An invalid channel id was used.

BadWindowId
: An invalid window id was used.

BadValueError
: An invalid or non-existent map product id was specified.

SEE ALSO

MAddProduct(3Map), MAddProducts(3Map), MChangeMap(3Map), MProdAtts(3Map), MRemoveProducts(3Map), MuReference(3Mu)

MRemoveProducts

FUNCTION

Remove specified list of map products from a geographc display.

SYNTAX

C Interface

```
void MRemoveProducts( channel, window, products, npro- ducts )
        Channel   channel;
        WindowId  window;
        ProductId *products;
        int  nproducts;
```

ARGUMENTS

channel — The connection to the Chart Manager; returned from MOpenChannel.

window — The window on which the products are currently displayed.

products — A list of map products to be removed from the map display.

nproducts — The number of products in the list.

DESCRIPTION

MRemoveProducts removes the specified products from the currently drawn geographic display. Products which are drawn on the current display are referenced using Produc- tIds. These values are returned as part of the product's MapProductAttributes structure.

ERRORS

AlreadyDrawingMap
: A map draw command is already in progress for the specified window.

BadChannel
: An invalid channel id was used.

BadWindowId
: An invalid window id was used.

BadValueError
: An invalid or non-existent map product id was specified.

SEE ALSO

MAddProduct(3Map), MAddProducts(3Map), MChangeMap(3Map),

MRemoveProduct(3Map), MuReference(3Mu)

MRemoveTimeOut

FUNCTION

Remove an interval timer.

SYNTAX

void MRemoveTimeOut(timer id)
MapTimerId timer id;

ARGUMENTS

timer id Specifies the Id for the timout request to be removed.

DESCRIPTION

The MRemoveTimeOut() routine removes a previously added time out. If the id is not a valid time out, no action is taken. Note that timeouts are automatically removed once they trigger. The MAddTimeOut() call returns a MapTimerId which can be used as an input to this function.

SEE ALSO

MAddTimeOut(3Map)

MRemoveVolume

FUNCTION

Remove a volume from the Map Search Path.

SYNTAX

C Interface

```
void MRemoveVolume( channel, volume )
        Channel  channel;
        char     *volume;
```

ARGUMENTS

channel	Specifies the connection to Cartographer; returned from MOpendChannel.
volume	The volume to be removed from the map search path.

DESCRIPTION

The MRemoveVolume removes a volume from the Map Search Path. All the maps that were present on this volume will no longer be available to the Client.

NOTE: Map products currently displayed on a map window whose source is the removed volume will have their rendering colors deallocated. On PseudoColor and GrayScale machines, this may unexpectedly change some map colors until the next map gets drawn.

ERRORS

BadChannel

An invalid channel id was used.

BadValueError

The specified volume is not part of the current map search path.

SEE ALSO

MReloadSearchPath(3Map), MAddVolume(3Map)

MReorderMaps

FUNCTION

Reorder specified map products on a geographic display.

SYNTAX

C Interface

```
void MReorderMaps( channel, window, products, nproducts )
    Channel   channel;
    WindowId  window;
    ProductId *products;
    int  nproducts;
```

ARGUMENTS

channel — The connection to the Chart Manager; returned from MOpenChannel.

window — The window on which the product is to be displayed.

products — The reordered list of map products to be displayed.

nproducts — The number of products in the list. This should match the number of products currently displayed.

DESCRIPTION

MReorderMaps redisplays the map products in the specified order. The map product listed first on the list will be displayed first, followed by the product listed second, and so forth. Hence the product listed last is effectively at the top of the display. Features and then objects are redisplayed once the reordering occurs.

ERRORS

AlreadyDrawingMap
: A map draw command is already in progress for the specified window.

BadChannel
: An invalid channel id was used.

BadWindowId
: An invalid window id was used.

BadValueError

An invalid or non-existent map product id was specified, or the total number of ids is not equal to the current number of displayed map products.

SEE ALSO

MAddProducts(3Map), MChangeMap(3Map), MProdAtts(3Map), MRemoveProducts(3Map), MuReference(3Mu)

MRequestFocus

FUNCTION

Request map focus.

SYNTAX

C Interface

```
void MRequestFocus ( channel, window )
        Channel     channel;
        WindowId    window;
```

ARGUMENTS

channel	Specifies the connection to Cartographer; returned from MOpenChannel.
window	The window whose focus is being requested.

DESCRIPTION

MRequestFocus requests that PointSelectEvents be sent to this process first. These events will only be sent if the process has its PointSelectMask set in its event mask. If no process requests focus, then the default is to have PointSelectEvents sent to the process which originally created the window.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

The window id passed is not valid.

BadOwner

A user requested focus on a window to which no connection has been made, or else a user request focus on a window where no PointSelect events have been requested.

SEE ALSO

MCreateMapWindow(3C), MOpenChannel(3C), MSendEvent(3C), MSetEventMask(3C), MReleaseFocus(3C)

MRestoreCursor

FUNCTION

Change cursor to the normal cursor.

SYNTAX

C Interface

```
void MRestoreCursor( channel, window )
        Channel  channel;
        WindowId window;
```

ARGUMENTS

channel	Specifies the connection to Cartographer; returned by MOpenChannel.
window	Specifies the window whose cursor state is to be set.

DESCRIPTION

The MRestoreCursor function restores the cursor to its normal state. The normal state is an object select state, with a cross hair cursor being displayed.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

An invalid window id was used.

SEE ALSO

MSetCursorMode(3C)

MScaleMap

FUNCTION

Rescale a map.

SYNTAX

C Interface

```
void MScaleMap( channel, window, scale factor )
Channel  channel;
WindowId window;
FLOAT    scale factor;
```

ARGUMENTS

channel — Specifies the connection to the Chart Manager; returned from MOpenChannel.

window — Specifies the window to be queried.

scale factor — A scale factor at which the map is to be redrawn. Value is relative to the currently displayed map. For example, a scale factor of 2.0 will result in coverage approximately twice as large as the current coverage, and a scale factor of 0.5 will result in coverage approximately half the current coverage.

DESCRIPTION

The MScaleMap function scales the currently displayed map using scale factor as a relative scaling value to the current map. The current map center, coloring and projection are all maintained. The product list used is the current list of products with the subtype field set to AnyMap. This enables map products in the same class to be automatically switched based on the current scale in effect.

ERRORS

This routine can generate just about the same set of errors as the MChangeMap command. Refer to that command for more information.

SEE ALSO

MChangeMap(3Map), MuReference(3Mu)

MSendEvent

FUNCTION

Send a map event to other users of a window.

SYNTAX

C Interface

```
void MSendEvent( channel, window, event, extra data ) Channel  channel;
                 WindowId window;
                 MapEvent *event;
                 Boolean  extra data;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

window — The window where the event occurs.

event — The event structure to send.

event size — The size fo the event structure.

extra data — Indiciates whether or not extra data is to be sent with the event. Set this value to True only if extra data is to be sent along with the event (see Note 2 below). Set this value to False otherwise.

DESCRIPTION

The MSendEvent function sends the specified event to other users of a map window. Only Client programs with MapUserEventMask set will receive these events. This can be used as a means for communicating information between processes connected to a common map window. The MapEvent structure is defined in MEvents.

Note 1: The MapEvents structure is not checked for validity, since there could be user defined events not defined in the MapEvents structure. The first five fields however, MUST, repeat MUST, match the MapGenericEvent structure, as all events are assumed to include these fields. The event type field should be set to a value greater than or equal to UserEvent.

Note 2: The sent event's size can be no larger than the size

of the MapEvent structure. This structure is a union of a number of other specific Cartographer Manager events. If your program wishes to pass a user event which is larger than this value, then it can be sent as extra data as follows:

a) Allocate space for the information to be sent. Set the extra data field in the GenericEvent structure to the address of this space. Set the extra bytes field in the GenericEvent structure to the size (in bytes) of the extra data to be sent.

b) Set the extra data parameter passed to the MSen- dEvent call to True.

Note 3: There is no provision currently for Clients using this mechanism. Indeed, the burden is on the Clients to ensure that user event types do NOT conflict with each other. Errors may result if the Client tries to send an event whose type is already defined by Cartographer (eg. value less than UserEvent).

ERRORS

BadChannel

The channel id is invalid.

BadWindowId

The window id is invalid.

BadValueError

The specified event type is invalid.

SEE ALSO

MEvents (3C)

MSetAnimateKeys

FUNCTION

Set the control keys used during animation.

SYNTAX

C Interface

```
#include <X11/keysym.h>
void MSetAnimateKeys( channel, window, select key, finish key, escape key )
    Channel   channel;
    WindowId  window;
    int  select key;
    int  finish key;
    int  escape key;
```

ARGUMENTS

channel	Specifies the connection to Cartographer; returned from MOpenChannel.
window	The id of the window whose attributes are to be set.
select key	The key to use for performing SELECT functions during animation.
finish key	The key to use for performing END functions during animation.
escape key	The key to use for performing ESCAPE functions during animation.

DESCRIPTION

MSetAnimateKeys() sets the current keys used during animation. It can be called at any time. Keys are set on a per window basis. The 3 keys which are specified are, in fact, X window Keysym's. Refer to the X window guides for more information on Keysym usage.

Each key takes on a default value if this routine is never called. The default key for select key is XK Return, which

generally maps to the "Return" key. The default key for finish key is XK KP Enter, which generally maps to the "Enter" key. The default key for escape key is XK Escape, which generally maps to the "Esc" key.

In order for the animation key sequence to work correctly, the three keys must each be unique. A BadValueError will occur if this is not the case.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

An invalid window was used.

BadValueError

An invalid key was specified.

SEE ALSO

MCreateObject(3C), MCreateText(3C), MModifyObject(3C)

MSetAttributes

FUNCTION

Set the attributes of an object.

SYNTAX

C Interface

```
void MSetAttributes( channel, object, atts, value mask )
        Channel   channel;
        ObjectId  object;
        ObjectAttributes    *atts;
        MapValueMask   value mask;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

object — Specifies the object for which the attributes will be set.

atts — Specifies the attributes of the object that will be set.

value mask — Specifies the value mask associated with the attributes that will be set.

DESCRIPTION

MSetAttributes sets the attributes of an object to those specified. Only the attributes with a corresponding bit in value mask are set. Only the attributes for the object itself are modified. If object is a List, then the attributes for the List itself are modified, but none of its children are affected. If object is a Class, then the attributes for the Class are modified, but the Class members remain unchanged until a call to MUpdateClass is made. If object is a Drawable, then the effect on the object is immediate. If object is a Template, then the template is redefined for future draw commands.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid object id was used.

BadValueError

One or more fields in atts contain invalid values.

SEE ALSO

MApplyAttributes(3C), MObjAtts(3C), MObjMask(3C), MSetColor(3C), MSetData(3C), MSetFillType(3C), MSetFont(3C), MSetLineStyle(3C), MSetLineType(3C), MSetPickability(3C), MSetVisibility(3C), MUpdateClass(3C)

MSetColor

FUNCTION

Change the color of an object.

SYNTAX

C Interface

```
void MSetColor( channel, object, color )
Channel   channel;
ObjectId  object;
char *color;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

object — The id of the object whose color is to be changed.

color — The new color for the object.

DESCRIPTION

MSetColor sets the color of a single object to a named color from the RGB database. Behaviorally, this call works in a similar manner to MSetAttributes(). The color must exist in the X Windows RGB Database. If it does not, the color is not changed, and an error message is generated.

ERRORS

BadChannel
: An invalid channel id was used.

BadObjectId
: An invalid object id was used.

BadValueError
: An invalid color name was specified.

SEE ALSO

MApplyColor(3C), MObjAtts(3C), MSetAttributes(3C),

MSetCursorAnnotation

FUNCTION

Set the annotation for the cursor, when in normal cursor mode.

SYNTAX

C Interface

```
void MSetCursorAnnotation( channel, window,
 range cursor, bearing cursor, calculation type,
value mask )
    Channel  channel;
    WindowId window;
    CursorState          range cursor;

    CursorState          bearing cursor;
    MapLineType          calculation type;
    MapValueMask  value mask;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned by MOpenChannel().

window — Specifies the window whose mode is to be set.

range cursor

Determines whether a circle will be displayed showing the range from the hook_point. Possible values are: CursorOn, CursorOff and CursorToggle.

bearing cursor

Determines whether a line will be displayed showing the bearing from the hook_point. Possible values are: CursorOn, CursorOff and CursorToggle.

calculation type

The method used to calculate disance, either GreatCircle or Rhumbline.

value mask

Indicates which values are to be set.

DESCRIPTION

The MSetCursorAnnotation() routine controls what is drawn on the cursor when the cursor is in NormalCursorMode. If either

range cursor or bearing cursor is set to CursorOn, a window appears at the bottom of the

map display, to the left of the cursor position readout. This window displays the current hook point, from which the range and bearing are calculated, as well as the current range and bearing from the hook point.

If bearing cursor is set to CursorOn, then a line is drawn on the screen from the hook point, to the current cursor location. If range cursor is CursorOn, then a circle is drawn centered around the hook point with the radius equal to the range from the hook_point to the current cursor location.

The value mask parameter determines which values for cursor annotation are to be set. Only the values with a corresponding bit in the value mask will be set. The constants CARangeCursor, CABearingCursor, and CACalculationType can be used to set the value mask.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

An invalid window id was used.

BadValueError

An invalid value was specified for cursor mode.

SEE ALSO

MQueryWindow(3C)

MSetCursorMode

FUNCTION

Set the mode of the cursor.

SYNTAX

C Interface

```
void MSetCursorMode( channel, window, cursor mode ) Channel  channel;
                       WindowId window;
                       MapCursor     cursor mode;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned by MOpenChannel().

window — Specifies the window whose mode is to be set.

cursor mode — The mode of the cursor.

DESCRIPTION

The MSetCursorMode() command sets the current mode of the cursor to the specified mode. The shape of the cursor will change, as well as the function of the left mouse button. The cursor will stay in the specified mode until the action is complete in Cartographer, or until another call to MSetCursorMode() is made with the cursor mode as MapNormal- Cursor. Some modes can only be exited by another call. All the modes are described below.

MapNormalCursor

This is the normal mode for the cursor, the cursor shape is a cross hair. Clicking on an object generates a ObjectSelectEvent to be sent Clicking on the map generates a PointSelectEvent.

MapWaitCursor

The cursor shape is a clock, and no selections are allowed on the map while in this mode. This mode can only be exited by a mode change typically from a call to MSetCursorMode() or MRestoreCursor().

MapHandCursor

This sets the cursor to Hand Pan mode. In this mode the cursor is shaped like a hand. The first click on the map attachs the map to the cursor. The map can then be moved about with the cursor. The second click

releases the map, and puts the cursor back into normal cursor mode.

MapRecenterCursor

This sets the cursor to Recenter mode. The shape of the cursor is a large dot. Clicking on a spot causes the map to recenter about this point. After one click, the mode returns to normal cursor.

MapZoomBoxCursor

This sets the cursor to Zoom Box mode. The shape of the cursor is four arrows pointing out. The first click selects a center point for a rubber band box. When released a rubber band box the same dimensions as the window is drawn on the map. The box will expand to the location of the cursor, retaining the aspect ratio of the window, until the mouse is clicked a second time. The map will then zoom in to the area selected, and the mode will return to normal cursor mode.

MapGroupBoxCursor

This sets the cursor to Group Box mode. The shape and actions of the cursor are the same as MapZoomBoxCursor, except that after the second click, a BoxEvent is sent back to the application program, as well as an ObjectSelectEvent for every object that lies within the box. The mode returns to normal cursor mode after the second click.

MapAreaSelectCursor

This sets the cursor to Area Select mode. The shape and actions of the cursor are the same as MapGroupBoxCursor, except that after the second click, a BoxEvent is sent back to the application, but no ObjectSelectEvents. This mode returns to normal cursor mode after the second click.

The animation keys can affect the cursor mode. The escape key will almost always place the cursor mode back to MapNormalCursor. The select key and finish key can be used in lieu of the input buttons on the input device for specifying points in various modes. You can modifiy these keys with the MSetAnimateKeys() call.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

An invalid window id was used.

BadValueError

An invalid value was specified for cursor mode.

SEE ALSO

MRestoreCursor(3Map), MSetAnimateKeys(3C)

MSetData

FUNCTION

Set the client data field of an object.

SYNTAX

C Interface

```
void MSetData( channel, object, client data )
Channel   channel;
ObjectId  object;
char *client data;
```

ARGUMENTS

channel — The connection to Cartographer; returned from MOpenChannel.

object — The id of the object whose client data attribute is to be set.

client data — The value to store in the client data field.

DESCRIPTION

MSetData sets the data field of an object to the specified value. Behaviorally, this call functions in a similar manner to MSetAttributes(). The client data field can be any 32 bit data item. This item is returned to the user when it is requested through MQueryObject, or by an ObjectSelectEvent.

ERRORS

BadChannel
: An invalid channel id was used.

BadObjectId
: An invalid object id was used.

SEE ALSO

MApplyData(3C), MObjAtts(3C), MSetAttributes(3C)

MSetEventHandler

FUNCTION

Set/modify map event handler.

SYNTAX

C Interface

```
void MSetEventHandler( handler )
MCallbackProc  handler;
```

handler — The name of the procedure to call whenever an event occurs.

DESCRIPTION

The MSetEventHandler specifies a procedure to be called when events from the Chart Manager are received by this Client. This procedure should be used in conjunction with MMainLoop(3Map). The callback procedure is expected to return void.

STRUCTURES

C Interface

```
typedef void *MCallbackProc();
```

SEE ALSO

MMainLoop(3Map), MAddInput(3Map), MAddTimeOut(3Map),

MSetEventMask

FUNCTION

Set/modify event mask.

SYNTAX

C Interface

```
void MSetEventMask( channel, window, event mask ) Channel  channel;
                      WindowId window;
                      MapValueMask  event mask;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

window — Specifies the window for which the event mask will be set/modified.

event mask — Specifies the new event mask. A mask of MapAl- lEventsMask indicates that all events will be received. A mask of MapNoEventsMask indicates that no events will be received.

DESCRIPTION

The MSetEventMask function specifies a new event mask. Events can be individually masked out based on a combination of the channel connection, the window, and the process connected to the channel.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

An invalid window id was used.

BadOwner

The window exists, but the current process has never connected to it.

SEE ALSO

MCreateMapWindow(3C), MEvents(3C), MEventMask(3C), MNextEvent(3C), MSendEvent(3C), MUseWindow(3C)

MSetFillOffset

FUNCTION

Set the pixel offset of an object and its children.

SYNTAX

C Interface

```
void MSetFillOffset( channel, object, fill offset ) Channel   channel;
        ObjectId  object;
        int  fill offset;
```

ARGUMENTS

channel — The connection to Cartographer; returned from MOpenChannel.

object — The id of the object whose fill offset is to be set.

fill offset — The new pixel fill offset for the object.

DESCRIPTION

MSetFillOffset sets the fill offset of an object and all of its children. Behaviorally, this call works in a manner similar to MSetAttributes(). This value provides a starting pixel offset for objects using FillTransparent fill type. Refer to MObjAtts(3C) for more information on this fields.

ERRORS

BadChannel
: An invalid channel id was used.

BadObjectId
: An invalid object id was used.

BadValueError
: An invalid fill offset value was specified.

SEE ALSO

MApplyFillOffset(3C), MObjAtts(3C), MSetAttributes(3C)

MSetFillType

FUNCTION

Set the fill type of an object.

SYNTAX

C Interface

```
void MSetFillType( channel, object, fill type )
Channel   channel;
ObjectId  object;
MapFillType    fill type;
```

ARGUMENTS

channel The connection to Cartographer; returned from MOpenChannel.

object The id of the object whose fill type is to be set.

fill type The new fill type of the object.

DESCRIPTION

MSetFillType sets the fill type of an object. Behaviorally, this routines functions in a similar manner to MSetAttributes(). Valid values for fill type are: FillEmpty, FillOpaque, FillTransparent, FillDotted, FillHorizontalStripes, FillVerticalStripes, FillNegativeSlants, FillPositiveSlants, and FillCrossHatch. If the fill type is FillOpaque, the object is filled completely with the color of the object. If the fill type is FillEmpty, only the border of the object is drawn. If the fill type is FillTransparent, then the object is filled with a pattern that allows the user to see the map through the object. The pattern which is drawn is a function of the fill weight and fill offset fields. Refer to MObjAtts(3C) for more information on these fields.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid object id was used.

BadValueError

An invalid fill type was specified.

SEE ALSO

MApplyFillType(3C), MObjAtts(3C), MSetAttributes(3C)

MSetFillWeight

FUNCTION

Set the fill weight of an object and its children.

SYNTAX

C Interface

```
void MSetFillWeight( channel, object, fill weight ) Channel   channel;
        ObjectId  object;
        int  fill weight;
```

ARGUMENTS

channel — The connection to Cartographer; returned from MOpenChannel.

object — The id of the object whose fill weight is to be set.

fill weight — The new fill weight of the object.

DESCRIPTION

MSetFillWeight sets the fill weight of an object and all of its children. Behaviorally, this call works in a manner similar to MSetAttributes(). The fill weight attribute is applied only when fill type is set to FillTransparent. Refer to MObjAtts(3C) for more information on how this works.

ERRORS

BadChannel
: An invalid channel id was used.

BadObjectId
: An invalid object id was used.

BadValueError
: An invalid fill weight was specified.

SEE ALSO

MApplyFillWeight(3C), MObjAtts(3C), MSetAttributes(3C)

MSetFont

FUNCTION

Set the font of an object.

SYNTAX

C Interface

```
void MSetFont( channel, object, font )
    Channel   channel;
    ObjectId  object;
    char *font;
```

ARGUMENTS

channel — The connection to Cartographer; returned from MOpenChannel.

object — The id of the object whose font is to be set.

font — The name of the new font for the object.

DESCRIPTION

MSetFont sets the object's font to the named font. Behaviorally, this routine works in a similar manner to MSetAttributes(). If the font name is not valid, the font remains unchanged, and an error message is issued. If object is not of type Text or Character, there is no real change as these are the only two object types that use the font parameter.

ERRORS

BadChannel
: An invalid channel id was used.

BadObjectId
: An invalid object id was used.

BadValueError
: An invalid font name was specified.

SEE ALSO

MApplyFont(3C), MObjAtts(3C), MSetAttributes(3C)

MSetHiLite

FUNCTION

Highlight/Unhighlight an object.

SYNTAX

C Interface

```
void MSetHiLite( channel, object, hilite )
Channel   channel;
ObjectId  object;
Boolean   hilite;
```

ARGUMENTS

channel — The connection to Cartographer; returned from MOpenChannel.

object — The object to be highlighted.

hilite — The highlight state. If set to True, then the object will be drawn in the current highlight color of the window. If set to False, the object will be unhilited, and will be displayed in its own color.

DESCRIPTION

MSetHiLite causes the object to be displayed in the current highlight color. Behaviorally, this function works in a manner similar to MSetAttributes(). If object is a List, then this command has no effect because a List is never actually drawn. The effects of this command on a Class object will only be seen if MUpdateClass() is called. The color of highlighted objects can be changed with the routine MSetHiLiteColor.

ERRORS

BadChannel
An invalid channel id was used.

BadWindowId
An invalid window id was used.

BadObjectId
An invalid object id was used.

SEE ALSO

MApplyHiLite(3C), MObjAtts(3C), MSetAttributes(3C), MUpdateClass(3C)

MSetHiLiteColor

FUNCTION

Set the color in which highlighted objects will be displayed.

SYNTAX

C Interface

```
void MSetHiLiteColor( channel, window, hilite color ) Channel  channel;
                      WindowId window;
                      char     *hilite color;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

window — The window whose highlight color is to be set.

hilite color — The new highlight color, must be a valid X Windows color name.

DESCRIPTION

MSetHiLiteColor sets the highlight color of the specified window to the named color. All objects that are currently highlighted will be changed to this color. The color name must be a valid name from the X Windows RGB Color Database.

ERRORS

BadChannel
: An invalid channel id was used.

BadWindowId
: An invalid window id was used.

BadValueError
: An invalid color name was specified.

SEE ALSO

MApplyHiLite(3C), MSetHiLite(3C)

MSetIntensity

FUNCTION

Set the color intensity on a geographic display.

SYNTAX

C Interface

```
void MSetIntensity( channel, window, intensity ) Channel     channel;
        WindowId  window;
        int  intensity;

void MSetIntensityDetail( channel, window, intensity, map, feature)
        Channel         channel;
        WindowId  window;
        int  intensity;
        MapProduct     *map;
        FeatureProduct *feature;

void MSetIntensityModels( channel, window, atts, natts) Channel    channel;
        WindowId  window;
                MapSetIntensityDetail    *atts;
        int  natts;

void MResetIntensity( channel, window)
        Channel         channel;
        WindowId  window;
```

ARGUMENTS

channel — The connection to the Chart Manager; returned from MOpenChannel.

window — The window where intensity is to be affected.

intensity — The new intensity of maps and features, range of 0-255.

map — The map product or products whose intensity is to be set.

feature — The feature product or products whose intensity is to be set.

atts — A list of detail intensity color models. See MColor(3Map) manual page for a description of this structure.

natts — The size of the list provided to MSetIntensityModels().

DESCRIPTION

ChartClients can modify the intensity of displayed maps and features in a map window to allow better viewing of objects and overlays, or to hi-lite certain maps or features versus others. Only maps and features which support the intensity color model (eg. allocate modifiable color resources when rendered) are affected by any of these calls, except when the item is rendered, at which point all maps and features are affected.

The MSetIntensity sets the intensity of all map products and features on a geographic display. The intensity is a value between 0 and 255. The intensity represents the maximum value for RGB values in the map or feature product's color map. If the value is outside of the valid range, it is forced to the end of the range. The call basically complies with previous versions of MSetIntensity() in that all maps and features will have this intensity. More detailed intensity settings are lost when this call is made.

The MResetIntensityModels call sets the intensity of all maps and features to the value specified by the last MSetIntensity call for this window, or equivalent. More detailed intensity settings are lost when this call is made.

The MSetIntensityDetail call loads a detailed intensity model for a given map window. Intensity models are sorted internally by Chart from most detailed to least detailed. When determining the intensity to set a map or feature, the first matching intensity model is used. For example, when

determining the intensity of a World Vector Shoreline map, a color model which specifies the intensity of map products {VectorMap, WorldVectorShoreLine}, will be used before a color model which specifies the intensity of map products {VectorMap, AnyMap}, which in turn would take priority over the general intensity model for map products {AnyMap, AnyMap}.

If the Chart Client is interested in affecting only maps, then specify a FeatureProduct with feature type and sub type fields set to NoFeature. Likewise, if the Chart Client is interested in affecting only features, then specify a MapProduct with map type and sub type fields set to NoMap.

Specifying a MapProduct with both fields set to AnyMap and a FeatureProduct with both fields set to AnyFeature is equivalent to the MSetIntensity() call.

If the intensity model already exists in the list for this window, then the existing model's intensity value is replaced by this model's intensity value, and all displayed maps and features which match this model are updated.

The MSetIntensityModels call acts similarly to the MSetIntensityDetail call, but this call allows a Chart Client to specify multiple intensity color models in the same call.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

An invalid window id was used.

BadValueError

An invalid range value was specified.

NOTES

Maps and features which are rendered with colors allocated in a degraded mode may see odd side effects, such as their intensity being modified by detailed intensity requests for other maps or features which should normally not effect them. This is because their colors are being shared with other maps and features. They may also see no effect when the intensity is changed because the colors are allocated read only. A Chart Client has no way of knowing this information beforehand. Other display hardware prevents intensity from working due to the way colors are allocated (such as TrueColor machines).

SEE ALSO

MChangeMap(3Map), MColor(3Map)

MSetLineStyle

FUNCTION

Set the line style for an object.

SYNTAX

C Interface

```
void MSetLineStyle( channel, object, line style ) Channel   channel;
                    ObjectId  object;
                    MapLineStyle   line style;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

object — The object whose line style is to be changed.

line style — The new line style.

DESCRIPTION

MSetLineStyle sets the line style attribute to the given value. Behaviorally, this function works in a manner similar to MSetAttributes(). Valid values for line style include: MapLineDashed, MapLineSolid, MapLineDotDashed, MapLineDotted, and MapLineDoubleDashed.

ERRORS

BadChannel
: An invalid channel id was used.

BadObjectId
: An invalid object id was used.

BadValueError
: An invalid line style was specified.

SEE ALSO

MApplyLineStyle(3C), MObjAtts(3C), MSetAttributes(3C)

MSetLineType

FUNCTION

Set the line type of an object.

SYNTAX

C Interface

```
void MSetLineType( channel, object, line type )
Channel   channel;
ObjectId  object;
MapLineType    line type;
```

ARGUMENTS

channel Specifies the connection to Cartographer; returned from MOpenChannel.

object The object whose line type is to be changed.

line type The new line type.

DESCRIPTION

MSetLineType specifies the line type attribute for an object. Behaviorally, this function works in a manner similar to MSetAttributes(). Valid values include: GreatCircle, RhumbLine, or GeoDesic.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid object id was used.

BadValueError

An invalid line type was specified.

SEE ALSO

MApplyLineType(3C), MObjAtts(3C), MSetAttributes(3C)

MSetLineWidth

FUNCTION

Set the line width of an object.

SYNTAX

C Interface

```
void MSetLineWidth( channel, object, line width ) Channel   channel;
        ObjectId  object;
        int  line width;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

object — The object whose line width is to be changed.

line width — The new line width.

DESCRIPTION

MSetLineWidth sets an object's line width attribute. Behaviorally, this function works in a manner similar to MSetAttributes(). The line width value is in pixels, and should not be less than 0.

ERRORS

BadChannel
: An invalid channel id was used.

BadObjectId
: An invalid object id was used.

BadValueError
: An invalid line width value was used.

SEE ALSO

MApplyLineWidth(3C), MObjAtts(3C), MSetAttributes(3C)

MSetMapBounds

FUNCTION

Set the geographic display boundary attributes.

SYNTAX

C Interface

```
void MSetMapBounds( channel, window, bounds )
 Channel  channel;
 WindowId window;
        MapBoundaryAttributes   *bounds;
```

ARGUMENTS

channel	Specifies the connection to the Chart Manager; returned from MOpenChannel.
window	Specifies the geographic display to be changed.
bounds	The new map boundaries to display. See MBoundary(3Map) for a full description.

DESCRIPTION

The MSetMapBounds command changes the geographic display coverage to the specified bounds. The projection, features, and color model remain unchanged. This call is identical to the MChangeMap call, with value mask set to CMSetBoundary. The map product list used is the current list of map products with the subtype field set to AnyMap. This enables map products in the same class to be automatically switched based on the scale currently in effect.

ERRORS

This routine generates the same set of errors as the MChangeMap command. Refer to that command for more information.

SEE ALSO

MChangeMap(3Map), MuReference(3Mu)

MSetMapColors

FUNCTION

Change the foreground and background colors in a geographic display.

SYNTAX

C Interface

```
void MSetMapColors( channel, window, background, fore- ground )
        Channel  channel;
        WindowId window;
        char     *background;
        char     *foreground;
```

ARGUMENTS

channel — Specifies the connection to Cartographer.

window — Specifies the window.

background — Specifies the name of the new background color.

foreground — Specifies the name of the new foreground color.

DESCRIPTION

The MSetMapColors function changes the foreground and background colors in a geographic display to the newly specified colors. The color names are character strings, and must be valid X Window RGB Database color names. The color names are used to search the RGB database to determine the actual RGB values for the specified colors.

Only those map and feature products which support the foreground/background color model will be affected by this call. This model is currently supported only by vector maps.

ERRORS

BadChannel
: An invalid channel id was used.

BadWindowId
: The window id passed is not valid.

BadValueError
: An invalid foreground or background color name was specified.

SEE ALSO

MCreateMapWindow(3C), MSetMapColorsByRGB(3C), MWindowAtts(3C),

MSetMapColorsByRGB

FUNCTION

Change the foreground and background colors in a geographic display.

SYNTAX

C Interface

```
void MSetMapColorsByRGB( channel, window,
                         bgnd red, bgnd green,
    bgnd blue,
                         fgnd red, fgnd green,
    fgnd blue )
        Channel  channel;
        WindowId window;
        unsigned short   bgnd red;

        unsigned short   bgnd green;
        unsigned short   bgnd blue;
        unsigned short   fgnd red;
        unsigned short   fgnd green;
        unsigned short   fgnd blue;
```

ARGUMENTS

channel Specifies the connection to Cartographer.

window Specifies the window.

bgnd red Specifies the red value for the background color.

bgnd green
Specifies the green value for the background color.

bgnd blue Specifies the blue value for the background color.

fgnd red Specifies the red value for the foreground color.

fgnd green
Specifies the green value for the foreground color.

fgnd blue Specifies the blue value for the foreground color.

DESCRIPTION

The MSetMapColorsByRGB function changes the colors of the background and

foreground in a geographic display to the newly specified colors. The colors are specified by providing the red, green, and blue values for the pixel. The values are actually 16 bit unsigned values as required by X Windows, with only the top eight bits actually used.

Only those map and feature products which support the foreground/background color model will be affected by this call. This model is currently supported only by vector maps.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

The window id passed is not valid.

SEE ALSO

MSetMapColors(3C), MWindowAtts(3C),

MSetMapWidth

FUNCTION

Set the geographic display width.

SYNTAX

C Interface

```
void MSetMapWidth( channel, window, map width )
  Channel  channel;
  WindowId window;
  FLOAT    map width;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

window — Specifies the window to be queried.

map width — The geographic display width at which the geographic display is to be drawn. Value is in nautical miles.

DESCRIPTION

The MSetMapWidth call sets the geographic display width to the specified value, and redraws it if necessary. This command scales the geographic display relative to the current width. The geographic display's center, projection, features, and color models do not change. The map product list used is the current list of map products with the subtype field set to AnyMap. This enables map products in the same class to be automatically switched based on the scale currently in effect.

ERRORS

SEE ALSO

MSetObjectData

FUNCTION

Set type-specific data for an object.

SYNTAX

C Interface

```
void MSetObjectData( channel, object, data, value mask )
        Channel   channel;
        ObjectId  object;
        ObjectData    *data;
        MapValueMask   value mask;
```

channel — The connection to Cartographer; returned from MOpenChannel().

object — The id of the object whose data attribute is to be set.

data — The type specific data to be used for the object.

value mask — A bit mask indicating which fields are to be set.

DESCRIPTION

MSetObjectData is used for modifying parts of an already existing object. The format of this routine is general purpose, intended to provide extensive editing capabilities for objects. In general, once an object is created, its type (eg. Drawable, List, Class, Template) and subtype (eg. Ellipse, Circle, etc.) cannot be changed. Hence, when using this call, care must be made to ensure that the values being changed are appropriate for the object. Normally, this information should first be obtained from MQueryObject(3C).

The MObjectData(3C) manual page describes the format of the data structure. Only those fields which have their bits set in the value mask will be modified. The value mask fields can take on the following values (note that not all values are valid for all objects):

Object Type, Value mask, Data element

Arc

OBDCenter	:	MapPoint	center
OBDMajorAxis	:	double	major_axis
OBDMinorAxis	:	double	minor_axis
OBDBearing	:	double	bearing
OBDAngle1	:	double	angle1
OBDAngle2	:	double	angle2

Bitmap

OBDLocation	:	MapPoint	location
OBDSetBitmap	:	int	width,height
	:	unsigned char	*bmap
OBDAddBitmap	:	unsigned char	*bmap

For OBDAddBitmap, the size of the bmap data array (width and height) must be the same as the original bitmap data. OBDXHot : int x_hot

OBDYHot	:	int	y_hot

XBitmap

OBDLocation	:	MapPoint	location
OBDSetBitmap	:	int	width,height
	:	Pixmap	xbitmap
OBDXHot	:	int	x_hot
OBDYHot	:	int	y_hot

XPixmap

OBDLocation	:	MapPoint	location
OBDSetPixmap	:	int	width,height
OBDXHot	:	int	x_hot
OBDYHot	:	int	y_hot
	:	Pixmap	xpixmap
	:	Pixmap	xbitmap

Box

OBDCenter	:	MapPoint	center
OBDWidth	:	double	width
OBDHeight	:	double	height
OBDBearing	:	double	bearing

Character

OBDLocation	:	MapPoint	location
OBDXOffset	:	double	x_offset
OBDYOffset	:	double	y_offset
OBDChar	:	char	c

Circle

OBDCenter	:	MapPoint	center
OBDRadius	:	double	radius

Ellipse

OBDCenter	:	MapPoint	center

OBDMajorAxis	:	double	major_axis
OBDMinorAxis	:	double	minor_axis
OBDBearing	:	double	bearing

Line

OBDPoint1	:	MapPoint	p1
OBDPoint2	:	MapPoint	p2
OBDPointOffset1	:	short	p1_x_offset, p1_y_offset
OBDPointOffset2	:	short	p2_x_offset, p2_y_offset

Polygon, Polyline

OBDPointList	:	MapPoint	*points
		int	npoints

Rectangle

OBDTopLeft	:	MapPoint	top_left
OBDBottomRight	:	MapPoint	bottom_right

Sector

OBDCenter	:	MapPoint	center
OBDRange1	:	double	range1
OBDRange2	:	double	range2
OBDBearing	:	double	bearing
OBDAngle1	:	double	angle1
OBDAngle2	:	double	angle2

Slash, Segment

OBDLocation	:	MapPoint	location
OBDBearing	:	double	bearing
OBDAngle1	:	double	angle
OBDLength	:	int	length

Symbol

OBDLocation	:	MapPoint	location
OBDSymbol	:	NTDSSymbol	symbol
OBDSize	:	int	point_size

Text

OBDLocation	:	MapPoint	location
OBDXOffset	:	double	x_offset
OBDYOffset	:	double	y_offset
OBDText	:	char	*text
		int	ntext

Weather

OBDPointList	:	MapPoint	*points
		int	npoints
OBDFront	:	FrontType	front_type

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid object id was used.

BadValueError

An invalid data field was used.

BadMaskValue

An invalid value_mask was used.

SEE ALSO

MObjectData(3C), MQueryObject(3C)

MSetOffset

FUNCTION

Modify a text object's offset.

SYNTAX

C Interface

```
void MSetOffset( channel, object, xoffset, yoffset ) Channel   channel;
        ObjectId  object;
        int  xoffset;
        int  yoffset;
```

ARGUMENTS

channel The connection to Cartographer; returned from MOpenChannel.

object The object whose text field is to be moved.

xoffset The horizontal offset in pixels. Positive values indicate points to the right of the object location, and negative values indicate points to the left.

yoffset The vertical offset in pixels. Positive values indicate points above the object location, and negative values indicate points below.

DESCRIPTION

MSetOffset modifies a SinglePoint object's offset values, without the overhead of destroying and creating the object. Using this command on non-single point objects will cause a BadValueError. Currently the following Drawable objects are SinglePoint objects: AngleText, Bitmap, Character, Charac- ter16, Segment, Slash, Symbol, and Text.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid object id was used.

BadValueError

The object is not a SinglePoint object.

SEE ALSO

FUTURE EXPANSIONS

The MSetOffset command is provided for compatability with earlier systems, and provides a simplified interface to the MSetObjectData() routine.

MSetPickability

FUNCTION

Set the pickability of an object.

SYNTAX

C Interface

```
void MSetPickability( channel, object, pickable ) Channel     channel;
                      ObjectId  object;
                      Boolean         pickable;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

object — The id of the object whose pickability is to be set.

pickable — The new pickability of the object. Valid values are Pickable, ParentPickable, and NotPickable

DESCRIPTION

MSetPickability sets the pickability of an object, and only that object. If object is a List or a Class, then the pickability is modified only on the ObjectAttributes for the List or Class object itself. You can affect the Class members by calling MUpdateClass.

The pickability determines whether or not an object is selectable on the window. In the simplest case, an object that is not a member of a List, if the object is selected and it's pickability is set to either Pickable or ParentPickable, then an ObjectSelectEvent is sent to the Cartographer Client that owns the object. If the object is not pickable, no event is sent.

If an object is a member of a List, and is Pickable, then an ObjectSelectEvent is sent to the Cartographer Client, just as if it were not in a List. If the object is not pickable, then no event is sent. If the object's pickability is set to ParentPickable then the pickability of the object's parent is checked. If the parent is pickable, then an ObjectSelectEvent is sent with the id of the List (not the id of the object actually selected). If the List is not pickable, no event is sent, and if the List is ParentPick- able, then this process continues until an object is found that is not pickable, or the top of the object tree is

reached. If the top List in the object tree is ParentPick- able, then an event is sent to the Cartographer Client just as if this object were set to Pickable.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid object id was used.

BadValueError

An invalid pickable parameter was specified.

SEE ALSO

MApplyPickability(3C), MEvents(3C), MObjAtts(3C), MUpdateClass(3C)

MSetPixel

FUNCTION

Change the color of an object.

SYNTAX

C Interface

```
void MSetPixel( channel, object, pixel )
Channel   channel;
ObjectId  object;
unsigned long  pixel;
```

ARGUMENTS

channel Specifies the connection to Cartographer; returned from MOpenChannel.

object The id of the object whose color is to be changed.

pixel The pixel value of the color for the object.

DESCRIPTION

MSetPixel sets the color of a single object to the specified pixel value. Behaviorally, this call works in a similar manner to MSetAttributes(). The pixel value must be a valid pixel value for the display. The validity of the color depends on the X Display hardware. Also Cartographer Clients must ensure that specified pixel colors are managed properly. The MSetColor is therefore recommended for most cases.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid object id was used.

BadValueError

An invalid color value was specified.

SEE ALSO

MApplyColor(3C), MObjAtts(3C), MSetAttributes(3C),

MSetPriority

FUNCTION

Set the pixel offset of an object and its children.

SYNTAX

C Interface

```
void MSetPriority( channel, object, priority )
Channel   channel;
ObjectId  object;
short     priority;
```

ARGUMENTS

channel — The connection to the Chart Manager; returned from MOpenChannel.

object — The id of the object whose fill offset is to be set.

priority — The new priority for the object.

DESCRIPTION

MSetPriority sets the priority of an object. The display will be updated to reflect the new stacking order of the objects. Behaviorally, this call works in a manner similar to MSetAttributes(). This value determines the order in which the objects are drawn on the screen. Refer to MObjAtts(3C) for more information on this field.

ERRORS

BadChannel
: An invalid channel id was used.

BadObjectId
: An invalid object id was used.

BadValueError
: An invalid fill offset value was specified.

SEE ALSO

MApplyPriority(3C), MObjAtts(3C), MSetAttributes(3C)

MSetSegment

FUNCTION

Change the bearing of a Drawable object of type Segment.

SYNTAX

C Interface

```
void MSetSegment( channel, object, bearing, length ) Channel    channel;
                  ObjectId  object;
                  FLOAT     bearing;
                  int  length;
```

ARGUMENTS

channel The connection to Cartographer; returned from MOpenChannel.

object The location to place the segment.

bearing The bearing of the segment in degrees.

length The length of the segment, in nautical miles.

DESCRIPTION

MSetSegment sets the bearing of the Segment object to the specified value. If the object is not a Segment drawable, no change is made, and an error message is issued.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid object id was used.

BadValueError

An invalid length was used.

SEE ALSO

MDrawSegment(3C)

MSetSymbolSize

FUNCTION

Modify the size of the drawn symbol in a symbol object.

SYNTAX

C Interface

```
void MSetSymbolSize( channel, object, size )
Channel   channel;
ObjectId  object;
int  size;
```

ARGUMENTS

channel — The connection to the Chart Manager; returned from MOpenChannel.

object — The object whose symbol is to be changed.

size — The new size of the symbol object. Supported values are: Tiny, Small, Medium, Large, Huge, TinyBold, SmallBold, MediumBold, LargeBold, and HugeBold.

DESCRIPTION

MSetSymbolSize changes the font size for NTDS symbol displayed in a symbol object, without the overhead of destroying and creating the object. Using this command on non-symbol objects will cause a BadValueError.

ERRORS

BadChannel
: An invalid channel id was used.

BadObjectId
: An invalid object id was used.

BadValueError
: The object is not a Symbol object.

SEE ALSO

MChangeSymbol(3C), MDrawSymbol(3C), MSetObjectData(3C)

FUTURE EXPANSIONS

The MSetSymbolSize command is a simplified interface to the MSetObjectData() function.

MSetTemplate

FUNCTION

Selectively copy one object's attributes to another object.

SYNTAX

C Interface

```
void MSetTemplate( channel, object, template,
    value mask )
        Channel   channel;
        ObjectId  object;
        ObjectId  template;
        MapValueMask   value mask;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

object — The object whose attributes are to be changed.

template — The object whose attributes are to be copied.

value mask — The bit mask representing those ObjectAttributes which are to be copied. See MObjMask(3C).

DESCRIPTION

MSetTemplate copies the attributes of a Template into an object. Only those attributes whose bit is set in value mask will be copied into the object. If object is a Drawable, then the ObjectAttributes from template are copied to the Drawable, and the graphics are updated. If object is a List, then the ObjectAttributes for template are copied to the List's ObjectAttributes, but are not copied to any of its children. Likewise, if object is a Class, then the ObjectAttributes for template are copied to the ObjectAttributes for the Class, but are not copied to any of the members of the Class. Use MUpdateClass to then update the members of the Class.

ERRORS

BadChannel
: An invalid channel id was used.

BadObjectId
: An invalid object id was used.

SEE ALSO

MApplyTemplate(3C), MObjAtts(3C), MSetAttributes(3C),
MUpdateClass(3C)

MSetVisibility

FUNCTION

Set the visibility of an object.

SYNTAX

C Interface

```
void MSetVisibility( channel, object, visibility ) Channel   channel;
                     ObjectId  object;
                     Boolean   visibility;
```

ARGUMENTS

channel The connection to Cartographer; returned from MOpenChannel.

object The id of the object whose visibility is to be set.

visibility The new visibility of the object.

DESCRIPTION

MSetVisibility sets the visibility of an object. If an active object's visibility is set to Hidden or False, then the object is not visible on the map. If an active object's visibility is set to Visible or True, then the object is visibile on the map. During the time in which an animated object is being created, but is not yet active, its visibility cannot be altered with client calls (refer to MCreateOb- ject()). If object is a List, then this routine has no effect as lists are never actually drawn. If object is a Drawable, then the effect on the object's visibility is immediate. If object is a Template, then the visibility state for the template is redefined. If object is a Class, then the visibility of all members of the class will only be affected upon calling MUpdateClass().

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

An invalid object id was used.

SEE ALSO

MApplyVisibility(3C), MObjAtts(3C), MSetAttributes(3C), MUpdateClass(3C)

MSync

FUNCTION

Flush output buffer, and wait until commands have executed.

SYNTAX

C Interface

```
void MSync( channel )
        Channel channel;
```

ARGUMENTS

channel	The connection to Cartographer; returned from MOpenChannel.

DESCRIPTION

MSync flushes the output buffer, and then waits until all requests have been received and processed by Cartographer.

ERRORS

BadChannel

An invalid channel id was used.

SEE ALSO

MOpenChannel(3C), MCloseChannel(3C)

MUnMapWindow

FUNCTION

Make a window invisible.

SYNTAX

C Interface

```
void MUnMapWindow( channel, window )
        Channel  channel;
        WindowId window;
```

ARGUMENTS

channel	Specifies the connection to Cartographer; returned from MOpenChannel.
window	Specifies the window to be made invisible.

DESCRIPTION

The MUnMapWindow removes a window in Cartographer from the screen. The window has not been destroyed, it is simply not visible.

ERRORS

BadChannel

An invalid channel id was used.

BadWindowId

An invalid window id was used.

SEE ALSO

MCreateMapWindow(3C), MMapWindow(3C), MDestroyWindow(3C)

MUpdateClass

FUNCTION

Updates the attributes of Class member objects.

SYNTAX

C Interface

```
void MUpdateClass( channel, class, value mask )
Channel  channel;
ObjectId class;
MapValueMask  value mask;
```

ARGUMENTS

channel — Specifies the connection to Cartographer; returned from MOpenChannel.

class — Specifies the object identifier of the Class to be updated.

value mask — Bit mask inidicating which fields in the Class are to be updated.

DESCRIPTION

MUpdateClass copies a Class object's ObjectAttributes into each of its member objects. Only the values whose bit is set in the bit mask will be updated. By using this command, several fields in a Class can be updated using the MSet... calls. The new ObjectAttributes can then be simulataneously applied to all of the class members using just one call.

This call also works with List objects in a similar manner to Class objects.

ERRORS

BadChannel

An invalid channel id was used.

BadObjectId

The specified object id is not a valid List or Class object.

SEE ALSO

MApplyAttributes(3C), MCreateClass(3C), MObjAtts(3C), MSetAttributes(3C)

MUseNamedWindow

FUNCTION

Register channel as a user of a window.

SYNTAX

C Interface

```
WindowId MUseNamedWindow( channel, wname, event mask ) Channel  channel;
                          char     *wname;
                          MapValueMask  event mask;
```

ARGUMENTS

channel — The connection to Cartographer; returned by MOpen-Channel.

wname — A pointer to the name of the window to connect to. This must match the window name field in the Win- dowAttributes structure for the MCreateMapWindow call. If more than one window has the same name, then the first window encountered with a matching name will be returned.

event mask — An initial mask of map events of interest to this process.

DESCRIPTION

MUseNamedWindow registers the specified channel as a user of the window. All events specified within the event mask will be sent to the application.

A Cartographer Client registered as a user of a window cannot destroy the window. Only the creator of the window is allowed to destroy the window. If the user makes a call to MDestroyWindow(), the call will be treated as an MReleaseWindow() call.

RETURN

If no errors occur, MUseNamedWindow will return the WindowId of the connected window. If the named window does not exist, or if it cannot be connected to, then the value InvalidWindowId is returned.

ERRORS

BadChannel
: The channel id was invalid.

BadWindowName
: No window with the specified name exists.

SEE ALSO

MCreateMapWindow(3C), MDestroyWindow(3C), MEvents(3C), MEventMask(3C), MReleaseWindow(3C), MUseWindow(3C)

MUseWindow

FUNCTION

Register channel as a user of a window.

SYNTAX

C Interface

WindowId MUseWindow(channel, window, event mask) Channel channel;
WindowId window;
MapValueMask event mask;

ARGUMENTS

channel — The connection to Cartographer; returned by MOpen-Channel.

window — The window to connect to. This value has been previously returned by an MCreateMapWindow call made by another Cartographer Client.

event mask — An initial mask of map events of interest to this process.

DESCRIPTION

MUseWindow registers the specified channel as a user of the window. All events specified within the event mask will be sent to the application.

This call differs from MUseNamedWindow only in the way in which it connects to a window (using the WindowId, instead of the window's name). Similar restrictions to MUseNamedWindow concerning releasing a window's connection apply to this call as well.

RETURN

The WindowId is returned as verification that the window exists and is accessible to the caller. The value InvalidWindowId is returned in the event that the specified value for window is invalid.

ERRORS

BadChannel
The channel id was invalid.

BadWindowId
The window id is invalid.

SEE ALSO

MDestroyWindow(3C), MEvents(3C), MEventMask(3C), MReleaseWindow(3C), MUseNamedWindow(3C)

MuAltitude

FUNCTION

Chart Manager altitude utility routines.

SYNTAX

C Interface

```
FLOAT MuTargetAltitude( altitude, range, angle, radius) FLOAT altitude;
    /*Input*/
    FLOAT range;    /*Input*/
    FLOAT angle;    /*Input*/
    FLOAT radius;   /*Input*/


FLOAT MuTargetMaxRange( altitude, target radius ) FLOAT altitude; /*Input*/
    FLOAT target;   /*Input*/
    FLOAT radius;   /*Input*/
```

ARGUMENTS

altitude The altitude of the view position.
range The range from the view position to the target.
angle View angle from the view position.

radius Effective radius of the earth.
target Altitude of the target.

DESCRIPTION

These three routines are unit independent. The return values will be in the units of the input variables. The radius of the earth can be:

EARTH RADIUS NM
EARTH RADIUS MI
EARTH RADIUS KM

RETURN VALUE

The MuTargetAltitude() function returns the altitude of the target in the respective units. The MuTargetMaxRange() function returns the maximum range that a target at the specified altitude can be seen.

MuConvert

FUNCTION

Chart Manager coordinate conversion routines.

SYNTAX

C Interface

```
int MuGeoMgr( geo, mgr)
    GEO_COORD  *geo;          /*Input*/

    MGR_COORD  *mgr;          /*Output*/

                int MuMgrGeo( mgr, geo)
    MGR_COORD  *mgr;          /*Input*/
    GEO_COORD  *geo;          /*Output*/


                int MuGeoUtm( geo, utm)
    GEO_COORD  *geo;          /*Input*/
    UTM_COORD  *utm;          /*Output*/

                int MuUtmGeo( utm, geo)
    UTM_COORD  *utm;          /*Input*/
    GEO_COORD  *geo;          /*Output*/


                int MuUtmMgr( utm, mgr)
    UTM_COORD  *utm;          /*Input*/
    MGR_COORD  *mgr;          /*Output*/

                int MuMgrUtm( mgr, utm)
    MGR_COORD  *mgr;          /*Input*/
    UTM_COORD  *utm;          /*Output*/

               Boolean MuValidMgr( mgr)
    MGR_COORD  *mgr;          /*Input*/

               Boolean MuValidUtm( utm)
    UTM_COORD  *utm;          /*Input*/

Boolean MuValidGeo( lng, lat)

    FLOAT lng;     /*Input*/
    FLOAT lat;     /*Input*/
```

ARGUMENTS

geo A geodetic coordinate structure.

utm A universal transverse mercator (UTM) coordinate.

mgr A Military Grid Reference (MGR) coordinate.

DESCRIPTION

MuConvert utilities provide a set of routines for converting between various map coordinate systems. The supported

systems include: Geodetic (GEO), Universal Transverse Mercator (UTM), and Military Grid Reference (MGR). The routines perform automatic validation checking of their inputs, and return CMNR SUCCESS if the conversion succeeds, and another status value if it fails. Failures in the routines will also place an invalid coordinate value in the result so that when used as inputs to other routines, failures propogate if the status is not checked. Routines where the output is a GEO coordinate will also normalize the result so that it falls between -180.0 and +180.0 degrees longitude, and -90.0 to +90.0 degrees latitude.

The MuValid...() routines check the validity of the input coordinate, and return True if the coordinate is valid, and False if the coordinate is invalid.

STRUCTURES

C Interface

```
typedef struct _MapPoint {

        double    lat;
        double    lon;
        double    alt;
} MapPoint, GEO_COORD;

                        typedef struct utm_coord {
double          easting;  /*Easting offset*/
double          northing; /*Northing offset*/
                    int  sphere;   /*UTM spheroid*/
int  zone;     /*UTM zone*/
                    } MapUTMPoint, UTM_COORD;

                        typedef struct mgr_coord {
        int         easting;  /*Easting offset*/
        int         northing; /*Northing offset*/
        short zone;     /*MGR world zone*/
char  grid_east;     /*Easting grid point*/ char  grid_north;    /*Northing grid point*/ char
grid_alpha;    /*MGR grid alpha*/ char  padding[3];    /*Unused padding*/

} MapMGRPoint, MGR_COORD;
```

The GEO COORD structure is used to represent a geodetic point on the earth. All input values are in degrees. The latitude field should lie between +84.0 degrees (north) and -80.0 degrees (south) for conversions to either UTM or MGR, as polar regions are not supported. The longitude field should lie between -180.0 degrees (west) and +180.0 degrees (east). Values outside this range are normalized.

The UTM COORD structure is used to represent a Universal Transverse Mercator point on the earth. The zone field represents the 6 degree longitudinal band on the earth for

this point. It should fall between -60 and +60. Use negative values for points in the Southern Hemisphere. The northing field must lie between 0.0 meters and 10 million meters. The easting field must lie within the span of a zone. The spheroid field specifies the spheroid model to use when performing transformations. The table below specifies the models which are supported, and the corresponding constant to use:

Spheroid Model	Field constant
Airy	SphAiry*
Australian National	SphAustralianNational
Bessel	SphBessel
Clarke 1866	SphClarke1866
Clarke 1880	SphClarke1880
Everest	SphEverest
International 1909	SphInternational1909
Modified Airy	SphModifiedAiry*
Modified Everest	SphModifiedEverest*
WGS 1972	SphWGS72

Those models marked with an asterisk are not supported for coordinate transformations from UTM to MGR.

The macro SphGuess(model)
may be used to provide a first guess for transforming a UTM coordinate into either an MGR coordinate or a geodetic coordinate. The transformation will revise the guess, and provide a more accurate spheroid model for the point if necessary, and return the actual spheroid used to the caller.

The MGR COORD structure is used to represent a Military Grid Reference point on the earth.
The zone field represents the 6 degree longitudinal band on the earth for this point. It should fall between +1 and +60.
Do not use negative values for points in the Southern Hemisphere.

The grid alpha field represents the latitude band for the point on the earth. Valid grid values are single, upper-case, letters between delineate the 100 sq. KM area of the earth which the point lies in. Extensive tables are searched during transformations to insure the validity of these values. Be sure to use upper-case letters. The easting and northing

fields are metric offsets between 0 and 99,999 from the lower left corner of the grid box.

RETURN STATUS

CMNR SUCCESS

Coordinate conversion successful - no errors.

CMNR BADMERIDIAN

Problems converting specified lat/long point to a UTM coordinate.

CMNR BADLAT

The specified latitude point is out of range. Valid range is -80.0 to 84.0.

CMNR BADLONG

The specified longitude point is out of range. Valid range is -180.0 to 180.0.

CMNR BADUTMZONE

The specified UTM zone is invalid. Valid range is 1 to 60.

CMNR BADUTMSPHERE

The specified UTM spheroid is invalid. Valid range is -10 to +10.

CMNR NOMGRGRID

The conversion algorithms failed when trying to find an MGR grid zone for the specified input GEO or UTM coordinate.

CMNR BADMGRZONE

The specified MGR zone is invalid. Valid range is 1 to 60.

CMNR BADMGRGRID

The specified MGR grid character is invalid. Valid range is 'A' thru 'X'. This character must be capitalized.

CMNR BADMGRCOORD

The specified MGR coordinate is invalid, out of range, or non-existent.

CMNR BADUTMNORTH

The specified UTM northing is out of range. Valid range is 0.0 (Nautical miles) to 10,000,000.0 Nautical miles.

CMNR BADUTM

The specified UTM coordinate is invalid, out of range, or non-existent.

MuDistance

FUNCTION

Chart Manager distance utility routines.

SYNTAX

C Interface

```
#include <M/Mconst.h>`
#include <M/Mtypes.h>

FLOAT MuBearing( p1, p2, gcrl)

    MapPoint        *p1; /*Input*/
    MapPoint        *p2; /*Input*/
    int   gcrl;     /*Input*/
FLOAT MuDistance ( p1, p2, gcrl) MapPoint       *p1; /*Input*/
    MapPoint        *p2; /*Input*/
    int   gcrl;     /*Input*/


void MuGetPosition( p1, bearing, range, gcrl, p2) MapPoint      *p1; /*Input*/
    FLOAT bearing;  /*Input*/ FLOAT range;    /*Input*/
    int   gcrl;     /*Input*/
    MapPoint        *p2; /*Output*/


void MuGetRangeBearing( p1, p2, gcrl, bearing, range) MapPoint  *p1; /*Input*/
    MapPoint        *p2; /*Input*/
    int   gcrl;     /*Input*/

    FLOAT *obearing;    /*Output*/ FLOAT *orange;  /*Output*/
```

ARGUMENTS

p1	A pointer to an input MapPoint structure which describes a geodetic position in degrees.
p2	A pointer to an input MapPoint structure which describes a second geodetic point in degrees.
gcrl	A flag which describes the type of line formulated between the input points. Two values are supported: GreatCircle, and RhumbLine. GreatCircle mode traces a great circle path between two

points, and in fact is the shortest distance between two points. RhumbLine mode calculates a straight line distance between two points, but may not be the shortest distance between two points.

bearing — A bearing value, in degrees, from the first input geodetic position p1.

range — A range value, in nautical miles, from the first input geodetic position p1, to the second point.

op2 — An output geodetic position, obtained by starting from point p1, and proceeding along a bearing of bearing degrees, for a range of range nautical miles.

obearing — An output bearing value, in degrees, obtained by determining the bearing between the input geodetic positions p1 and p2.

orange — An output range value, in nautical miles, obtained by determining the distance between the input geodetic positions p1 and p2.

DESCRIPTION

The MuBearing() routine returns a bearing value in degrees given two input geodetic positions points p1 and p2. The MuDistance() routine returns a distance value in nautical miles given two input geodetic positions. The gcrl flag indicates the type of route traversed between the two points. A value of GreatCircle will return the great circle traversal, and is the shortest distance between two geodetic points. A value of RhumbLine will return the linear (bearing/range) traversal distance.

The MuGetPosition() routine returns a geodetic position given an input point p1, and a bearing and range value. The MuGetRangeBearing() routine returns a bearing and range, given two input geodetic points p1 and p2.

RETURN

MuDistance() returns the distance between the two points in Nautical Miles. MuBearing() returns the bearing between two points in degrees.

MuError

FUNCTION

Chart Manager standardized error and warning utilities.

SYNTAX

C Interface

```
int MuAppError( format [ , arg... ])
        char  *format;  /*Input*/

int MuSysError( format [ , arg... ])
        char  *format;  /*Input*/

int MuAppWarning( format [ , arg... ])
        char  *format;  /*Input*/

int MuSysWarning( format [ , arg... ])
        char  *format;  /*Input*/

int MuErrorHandler( channel, window, major code, minor code,
          error code, error info)
        Channel    channel;
        WindowId   window;
        Protocol   major code;

                        Protocol   minor code;
        MapStatus  error code; MapErrorCodeInformation   *error info;


#include <M/MuError.h>

                void MuErrorMsg( code [, arg... ])
        ErrorCodeType   code;

void MuSetErrorList( list) ErrorCrossReference  *list;
```

ARGUMENTS

format — Any format string accepted by printf(3).

channel — The connection to the Chart Manager; returned from MOpenChannel(3C).

window — The window where the error occurred.

major code

The major code number of the generated error.

minor code

The minor code code number of the generated error.

error code

The code number of the generated error. Error codes are described in each Chart Manager M library manual page under the "ERRORS" heading. See also MError(3C) for a synopsis.

error info

Additional information specified with some errors (specifically map related errors). See MError(3C) and MChangeMap(3Map).

code — A special error code designator. Some error code designators have one to one correspondence with error status codes returned by the Chart Manager. Additional error codes are defined within the Mu library to specify additional commonly occuring errors. The programmer can also define an additional set of error codes. The core set of codes defined by the Mu library is described in the VALUES section of this manual page.

list — A cross reference list containing error codes and error messages. The core set of error codes is described in the VALUES section of this manual page. The programmer can also specify an additional set of error codes for a program.

class — The classification of an error code. Allowed values are: WARNING, ERROR, SEVERE, FATAL, INFOR- MATION, CONSOLE MSG, and SUCCESS.

DESCRIPTION

The Mu library provides a rather extensive set of error processing routines principally designed to provide Chart Clients with a consistent means of handling errors which originate from requests to the Chart Manager. As a secondary feature, additionally error support is provided for exceptions which commonly occur (such as file access problems, improperly entered data, etc).

The routines MuAppError(), MuAppWarning(), MuSysError(), and MuSysWarning() are simply routines which process client errors, warnings, system errors, and system warnings in a consistent manner.

The MuErrorHandler() routine is a Chart Client error handler routine which can serve as an input to MSetErrorHandler(3Map).This Chart Manager error codes, in particular the

rather involved codes returned by geographic display-related requests.

The MuErrorMsg() routine interprets both Chart Client MapStatus codes, and additional error codes to provide a core set of error support to a Chart Client. The VALUES section below describes the error codes supported by the Mu library. The Chart Client can define additional error codes using the MuSetErrorList() call. The ErrorCodeType value associates both an error status code, and an error class into one value. The class allows the Mu library to process errors in distinct classes differently. For example, FATAL errors result in the Chart Client process being terminated, whereas INFORMATION messages allow the Chart Client to continue processing.

All Mu library error processing calls the program $PROGS/Warning, if it exists. This results in a formatted message being displayed in a X Window. If the Warning utility is not found the message is printed to stderr. Extra long messages are automatically broken up onto separate lines, although carriage returns using \n are recognized inside of error messages.

MuAppError() prints out the specified message, preceded by the string "Error - ". MuAppWarning() prints out the specified message, preceded by the string "Warning - ".

MuSysError() prints out the specified message, preceded by the string "Error - ", and followed by the system error message set by errno (see intro(3)). MuSysWarning() works similarly, except that the specified message is preceded by the string "Warning - ".

RETURN

MuAppError(), MuAppWarning(), MuSysError(), and MuSysWarning() return -1 if the call fails. Otherwise the number of arguments which were printed out as part of the message is returned.

MuErrorHandler() returns 0 if the error was processed, and -1 otherwise. When invoked using the MSetErrorHandler() call this allows for default processing of error codes not processed by MuErrorHandler().

STRUCTURES

C Interface

```
typedef int ErrorCodeType;
typedef struct {
        ErrorCodeType   code;
        char  *string;
} ErrorCrossReference;
```

An ErrorCodeType value consists of both an error code value and an error class value.

These values are bitwise OR'd together to form the ErrorCodeType. Valid error codes defined by the Mu library are described in the VALUES section of this manual page. When defining your own error codes, follow the syntax shown in the file MuError.h. Bear in mind that code values between 0x0000 and 0x1fff are reserved by the Mu library.

The ErrorCrossReference structure is used to define additional Chart Client error codes. The structure consists of simply an ErrorCodeType value, and a corresponding error message. The error message should be in the format suitable for printf(3).When MuErrorMsg() is invoked with the Error- CodeType value set to this error code, then sufficient arguments should be included to fill in any values in the error message string.

VALUES

The following error code values are recognized by the MuEr- rorMsg routine. For the specific error messages, refer to the file MuError.h.

Error code value	Default error class	Parameters
MErrAmbKey	ERROR	Key word (%s)
MErrBadCoord	ERROR	Geodetic coordinate specified (%s)
MErrBadNumber	ERROR	Specified number (%s)
MErrBadVersion	SEVERE	Filename(%s), version(%f), running version(%f)
MerrBogusValue	INFORMATION	Value name(%s), value (%d)
MErrCValueRange	ERROR	Name(%s),value(%s),lower range(%c),upper range(%c)
MErrDValueRange	ERROR	Name(%s),value(%s),lower range(%d),upper range(%d)
MErrEndOfFile	WARNING	Filename(%s)
MErrFileOpen	ERROR	Filename(%s)
MErrFileRead	ERROR	Bytes expected to read(%d), bytes read(%d)
MErrFileWrite	ERROR	Bytes expected to write(%d), bytes written(%d)
MErrFValueRange	ERROR	Value name(%s), value(%s), lower range(%f), upper range(%f)
MErrGeneral	ERROR	Any error message. Parameters vary.
MErrIllKey	ERROR	Key word (%s)
MErrNameExists	ERROR	Value name (%s), Key word (%s)
MErrNoEnvDef	ERROR	Environment variable(%s)
MErrNoMaps	WARNING	Map search path (%s)
MErrNoName	ERROR	None
MErrNoSuchName	ERROR	Value name (%s), key word (%s)

MErrPrintFail	ERROR	Job name (%s), printer name (%s)
MErrPrinted	SUCCESS	Job name (%s), printer name (%s)
MErrRecDeleted	SUCCESS	Key word (%s)
MErrRecNoDelete	ERROR	Key word (%s)
MErrSelectNone	ERROR	None
MErrSelectOne	ERROR	None
MErrXValueRange	ERROR	Name(%s),value(%s),lower range(%x),upper range(%x)
MerrNoSuchMsg	SEVERE	Undefined message code (reserved)

The following error codes correspond directly to status

codes received from the Chart Manager. All of these codes expect three parameters: a string which says "Map Product" or "Feature Product", the product type string, and the projection string. Note: all other MapStatus codes result in the MErrNoSuchMsg error message being printed.

Error code value	Default error class
AlreadyDrawingMap	INFORMATION
BadMapEntry	ERROR
BadServer	CONSOLE MSG
BadValueError	CONSOLE MSG
ErrorDrawingFeature	WARNING
ErrorDrawingMap	WARNING
FeatureNotAvailable	INFORMATION
FeatureNotSupported	ERROR
MapDrawAborted	INFORMATION
MapTooSmall	CONSOLE MSG
MaxExtents	CONSOLE MSG
MaxScale	ERROR
MinScale	ERROR
NoMapsDrawn	INFORMATION
ProductNotFound	WARNING
ProjectionNotSupported	ERROR
SystemNotSupported	ERROR
TooManyMaps	WARNING
UnresponsiveDrawModule	WARNING
WorldFitProblem	CONSOLE MSG

The following error classes are recognized by the MuEr-

rorMsg() call.

Error class value	Chart Client action
CONSOLE_MSG	Prints a message and continue processing
ERROR	Pause until user confirms reading error message
FATAL	Terminate Chart Client
INFORMATION	Continue
SEVERE	Pause until user confirms reading error message
SUCCESS	Continue
WARNING	Continue

EXAMPLES

```
#define CClientBadMGRCoord    (0x2000 | ERROR)

#define CClientBadUTMCoord    (0x2001 | ERROR) ErrorCrossReference ref[] = {

        {CClientBadMGRCoord, "The value %s is an improperly formatted MGR
        coordinate"}, {CClientBadUTMCoord, "The value %s is an improperly formatted
        UTM coordinate"}, {0, NULL}};

MuSetErrorList (ref);
MSetErrorHandler (MuErrorHandler);
MuErrorMsg (MErrFileOpen, "myfile.dat");
MuErrorMsg (MErrGeneral, "Please specify a file name other than %s", filename);
MuErrorMsg (CClientBadMGRCoord, mgr_string);
MuAppError ("Do not specify %s for a map name!", map_name);
```

ENVIRONMENT

PROGS

Location of Chart Manager executables.

FILES

Warning

If this file is present, then a warning window will be

produced. Otherwise, the message is printed to stderr. For the CONSOLE MSG error class, the message is always printed to stderr.

SEE ALSO

MError(3C), MuInit(3Mu), stdio(3), printf(3V)

MuFont

FUNCTION

Convenvience routines for accessing predefined NTDS fonts.

SYNTAX

C Interface

```
char *MuPointSizeToFontName( size )
        int   size;

int MuPointSizeToWidth( size )
        int   size;
```

ARGUMENTS

size	The constant point size name for the font, e.g. Small.

DESCRIPTION

These routines return specific information about the predefined NTDS fonts. MuPointSizeToFontName() returns the name of the font representing the specified point size. The string returned should not be modified. MuPointSizeToWidth() returns the width in pixels of the specified font.

RETURN

MuPointSizeToFontName() returns the full X window font specification given the input size. This string value is statically assigned, and should NOT be freed using MFree. The value NULL is returned if the input size is invalid. MuPointSizeToWidth() returns the width of each symbol in the specified font (in pixels). The value 0 is returned if the input size is invalid.

SEE ALSO

MDrawSymbol(3C)

MuGeoPosn

FUNCTION

Chart Manager geo reference position string utility routines.

SYNTAX

C Interface

```
void MuGeoToString( string, longitiude, latitude) char  *string;  /*Output*/

                                FLOAT longitude;     /*Input*/
                                  FLOAT latitude; /*Input*/

int MuStringToGeo( string, latitude, longitude) char  *string;  /*Input*/
        FLOAT *latitude;     /*Output*/

        FLOAT *longitude;    /*Output*/
```

ARGUMENTS

string The string representation of a geodetic reference point. The format of this string is:

<clon><clat> [<sslon> [<sslat>]]

where

<clon> is a 2 character longitude grid value <clat> is a 2 character latitude grid value <sslon> is a 2 digit longitude seconds value (0 to 59)
<sslat> is a 2 digit latitude seconds value (0 to 59)

latitude The latitude value in degrees.

longitude The longitude value in degrees.

DESCRIPTION

The MuGeoPosn routines provide conversions from geodetic values to a printable string, and vice-versa. The format of the string is in the geo-reference grid format, which divides the world into 24 latitude and 24 longitude bands, with character designations for each band. Each band in turn is subdivided into 24 sub-bands, and an optional seconds field for additional precision.

The routine MuGeoToString() returns a string representation of the input position. The parameter string must be large enough to accept the output (at least 9 characters). The input coordinate is normalized to always fall between -180.0 and 180.0 degrees longitude, and -90.0 to 90.0 degrees latitude. Invalid inputs will result in a string of asterisks.

The routine MuStringToGeo() parses the input string and returns a latitude value and a longitude value if the

position string is valid.

RETURN

MuStringToGeo() returns the value CMNR SUCCESS if the conversion is successful, and returns CMNR BADFORMAT if the conversion fails. In the event of failure, the returned coordinate is set to (+infinity, +infinity).

MuInit

FUNCTION

Chart Client Initialization Functions.

SYNTAX

C Interface

```
#include <X11/Xlib.h>
#include <M/Mlib.h>
#include <M/Qualifiers.h>
#include <M/Mu.h>
#include <M/MuInit.h>

int MuInitialize( argc, argv, inatts, inmask, outatts) int   argc;     /*Input*/
      char  **argv;   /*Input*/
      ClientInputAttributes     *inatts;  /*Input*/ MapValueMask    inmask;
      /*Input*/ ClientOutputAttributes    *outatts; /*Output*/

void MuTerminate( )
```

ARGUMENTS

argc — A count of the number of arguments passed to the Chart Client when the command is invoked.

argv — The NULL terminated list of command line arguments.

inatts — A structure which describes special initialization attributes over and above the default attributes used to initialize the Chart Client's connection to the Chart Manager.

inmask — A bit mask which indicates those fields in inatts which are being specified. Those fields not specified imply certain default actions.

outatts — A structure containing key fields and values for communicating with the Chart Manager, and to a connected window.

SEE ALSO

MAddInput(3Map), MAddTimeOut(3Map), MError(3C), MEventMask(3C), MMainLoop(3Map), MOpenChannel(3C), MSetEventHandler(3Map), MUseNamedWindow(3C), MUseWindow(3C), MuError(3Mu), MuOption(3Mu), XOpenDisplay(3X11), execv(3), signal(3)

MuMgrPosn

FUNCTION

Chart Manager MGR position string utility routines.

SYNTAX

C Interface

void MuMgrToString (string, mgr)

char *string; /*Output*/ MGR_COORD *mgr; /*Input*/

int MuStringToMgr (string, mgr) char *string; /*Input*/ MGR_COORD *mgr; /*Output*/

ARGUMENTS

string The string representation of a Military Grid Reference System (MGRS) position. The syntax of this string is:

<zone> <alpha> <grid-east><grid-north>[]<easting>[-]<northing>

where

<zone> is the Universal Transverse Mercator (UTM) zone.
<alpha> is the UTM grid alpha character. <grid-east> is the MGRS easting grid location character.
<grid-north> is the MGRS northing grid location character.
<easting> is the MGRS easting offset value in meters. Valid range is between 0 and 99999 in increments of 1 meter.
<northing> is the MGRS northing offset value in meters. Valid range is between 0 and 99999 in increments of 1 meter.

mgr The MGRS coordinate record.

DESCRIPTION

The MuMgrPosn routines provide conversions from MGR COORD records to a printable string, and vice-versa. The routine MuMgrToString() returns a string representation of the input position. The parameter string must be large enough to accept the output (at least 19 characters).

The routine MuStringToMgr() parses the input string and returns an MGRS record value if the position string is valid. The MuStringToMgr() routine calls MuMgrGeo() to validate the input position. An invalid MGR coordinate is guaranteed to be returned if the MuStringToMgr() call fails.

Likewise, an invalid MGR coordinate will return an asterisk string in the

MuMgrToString() call.

RETURN

The MuStringToMgr() routine returns the value CMNR SUCCESS if the conversion is successful, and returns one of a number of possible error codes if the conversion fails. The error codes are explained in detail in the MuConvert(3Mu) manual page.

SEE ALSO

MuConvert(3Mu)

MuOption

FUNCTION

Chart Manager option parsing routines.

SYNTAX

C Interface

```
#include <M/Qualifiers.h>
        int MuQualGetOption( options, args, count,
extra values)
    Qualifiers *options;   /*Input*/

    char  **args;   /*Input*/ int   *count;    /*Output*/
char  ***extra values;     /*Output*/

char *MuQualGetPgmName( )

void MuQualUsage( name, options)

    char  *name;    /*Input*/
    Qualifiers *options; /*Input*/
```

ARGUMENTS

options — A pointer to a list of Qualifiers structures. Each record describes a supported option for this command, and indicates whether a parameter follows.

args — A pointer to a list of strings which specify the list of input arguments. This list must be terminated by a NULL pointing string.

count — A modifiable offset, initially set to the offset into the string list where parsing should begin, and modified by the MuQualGetOption() routine as options get parsed.

extra values — A pointer to a list of command lines values which are not tied to an option. These values will always follow all options on the line, and allow command syntax such as "command -option <value> <file1> <file2> ...".

name — The command name, in string format.

DESCRIPTION

The MuQualGetOption() routine returns the next matching option in the command line.

Options are specified using a list of Qualifiers records, each of which contains the option value, and a modifier. The Qualifiers structure is described in the STRUCTURES section below.

This routine returns -1 whenever an error occurs during parsing. When an option matches, the index of the matching option is returned. When an extra parameter is encountered, MuQualGetOption() returns the value of the highest matching option index, plus 1. When all arguments have been checked, the routine returns -2. At most one parameter is supported per option.

The MuQualGetOption() routine should be called successively until the value -2 is returned, which means that there are no more options to be parsed. When an option is successfully parsed, the index of the option is returned by this routine, and the option's value (if any) is placed in the specified address field in the Qualifiers structure. If no more options exist on the line, but additional command line values do exist, then the value equal to the number of Qual- ifiers records + 1, is returned, and the parameter extra values is set to the address of the first argv containing the command line values. In the event that -1 is returned, the caller should most likely print out the command syntax, using MuQualUsage(), and exit. When errors occur, MuQualGetOption() generally prints out an explanation as to the cause.

The MuQualGetPgmName() routine conveniently removes any preceding path name from a command line. Generally speaking, the input to this routine should be the value argv[0].

The MuQualUsage() routine conveniently prints out the syntax for the given command. This routine requires the command name, as well as a list of Qualifiers structures.

RETURN

The MuQualGetPgmName() routine returns a character string containing the name of the executable. This value is statically assigned and should NOT be freed using MFree(3Map).

The MuQualGetOption() routine returns the index of the next option on the command line. The value -1 is returned whenever an invalid option, or other parsing error occurs. The value -2 is returned after the entire command line has been checked.

STRUCTURES

C Interface

```
typedef enum {
        PARAMETER, OPTIONAL_PARAMETER, NOPARAMETER
} ParmRequirements;

typedef struct {
        char *option;
```

```
        ParmRequirements optiontype;
        char **value;
} Qualifiers;
```

The fields for the Qualifiers structure are defined as follows:

option

A pointer to a string representing the option.

optiontype

The syntax supported. The value PARAMETER means that a parameter is required, and follows. The value NOPARAM- ETER means that no parameter is required. The value OPTIONAL PARAMETER means that a parameter may follow, but is not required. In this case, a parameter is not parsed if the next item specified is an option, or else is NULL.

value

An address to place an option's value. This should be an address of a pointer to a string. Since the returned value is one of the argv's, it does not need to allocate space for a string, since argv pointers are globally defined. This value applicable only if optiontype is set to PARAMETER.

MuPosition

FUNCTION

Chart Manager geodetic position string utility routines.

SYNTAX

C Interface

```
void MuPositionToString( string, longitude, latitude, format )
    char  *string;    /*Output*/

    FLOAT longitude;      /*Input*/ FLOAT latitude;      /*Input*/ int   format;

int MuStringToPosition( string, longitude, latitude) char  *string;        /*Input*/

                FLOAT *longitude;       /*Output*/
                FLOAT *latitude;        /*Output*/
```

ARGUMENTS

string The string representation of a latitude/longitude combination. In the routine MuStringToPosition(), the format of this string is:

<dd>:[<mm>:[<ss>]][EWNS] <ddd>:[<mm>:[<ss>]][EWNS]

where

<dd> is the whole number of degrees.

<mm> is the number of minutes (0 to 59). <ss> is the number of seconds (0 to 59).

[NS] is the latitude hemisphere (N: Northern, S: Southern).

[EW] is the longitude hemisphere (E: Eastern, W: Western).

DESCRIPTION

The MuPosition routines provide conversions from geodetic values to a printable string, and vice-versa. The routine MuPositionToString() returns a string representation of the input position. The parameter string must be large enough to accept the output (at least 20 characters). The input latitude is normalized to fall between -90 and +90 degrees, and the input longitude is normalized to fall between -180 and +180 degrees. An invalid input position will result in a formatted asterisk string.

The routine MuStringToPosition() parses the input string and returns a latitude value and a longitude value if the position string is valid. In the event of an error, MuStringTo- Position() returns a coordinate value of (+infinity, +infin- ity).

RETURN

The MuStringToPosition() routine returns the value CMNR SUCCESS if the conversion is successful, and returns one of the following values if the conversion fails: CMNR BADFORMAT, CMNR BADLAT, or CMNR BADLONG.

SEE ALSO

MuConvert(3Mu), MuMgrPosn(3Mu), MuUtmPosn(3Mu)

BUGS

The MuPositionToString() routine should accept a string format and length as parameters. This would make the routine more general purpose. The MuStringToPosition() should also accept a format string for parsing the input location string.

MuReference

FUNCTION

Chart Manager category reference routines. Most of these functions operate on MapReference lists, which are first retrieved using "List" functions. For example, MuListTypes returns a pointer to a list of type MapReference and the size of the list is returned via the argument list. Subsequently, the list and its size may be used in to call functions such as MuStringToType.

SYNTAX

C Interface

```
MapReference *MuListTypes( listsize)

        int   *listsize;
MapType MuStringToType( string, list, size) char  *string;
          MapReference          *list;
          int       size;

char *MuTypeToAcronym( value, list, size) unsigned int      value;

          MapReference          *list;
          int       size;

char *MuTypeToString( value, list, size) unsigned int          value;
          MapReference          *list;
          int       size;

int MuAddType( record)

        MapReference            *record;


unsigned int MuStringToSubType( string, list, size) char  *string;
          MapReference          *list;
          int       size;

char *MuSubTypeToAcronym( value, filter, list, size) unsigned int  value;

        unsigned int            filter;
          MapReference          *list;
          int       list;

char *MuSubTypeToString( value, filter, list, size) unsigned int       value;
```

```
        unsigned int                filter;
          MapReference          *list;
          int       size;
MapStatus MuAddSubType( record)


        MapReference   *record;
              MapReference *MuListSubTypes( filter, size)
        unsigned int       filter;
        int   *size;


unsigned int MuStringToProjection( string, list, size) char  *string;
          MapReference          *list;
          int                   size;
            char *MuProjectionToAcronym( value, list, size)
        unsigned int       category;

          MapReference          *list;
          int                   size;

char *MuProjectionToString( value, list, size) unsigned int   value;
          MapReference          *list;
          int                   size;

int MuAddProjection( record)
        MapReference   *record;



              MapReference *MuListProjections( listsize)
        int   *listsize;

unsigned int MuStringToFeature( string, list, size) char  *string;
          MapReference          *list;
          int                   size;

char *MuFeatureToAcronym( value, list, size) unsigned int  value;

          MapReference          *list;
          int                   size;

char *MuFeatureToString( value, list, size) unsigned int       value;
          MapReference          *list;
          int                   size;

int MuAddFeature( record)
```

MapReference *record;

MapReference *MuListFeatures(listsize)
int *listsize;

unsigned int MuStringToSubFeature(string, list, size)

char *string;
MapReference *list;
int size;

char *MuSubFeatureToAcronym(value, filter, list, size) unsigned int value;

unsigned int filter;
MapReference *list;
int size;

char *MuSubFeatureToString(value, filter, list, size) unsigned int value;
unsigned int filter;
MapReference *list;
int size;

int MuAddSubFeature(record)
MapReference *record;

MapReference *MuListSubFeatures(filter, size)
unsigned int filter;
int *size;

ARGUMENTS

string A valid string representation of the particular category. The string can be all or part of a category's string representation, acronym representation, or a combination of both representations. In the latter case, the accepted format is "string (acronym)". The input string must be long enough to unambiguously delineate a unique category. Otherwise the category UnknownValue is returned.

category The internal representation for the particular category.

type An internal representation for a particular

category. See DESCRIPTION section below for more information.

list — The cross reference list for a particular cross reference item. Returned by the corresponding MuList... call for that item.

size — The size of the cross reference list. Returned as a parameter by the corresponding MuList... call.

record — A MapReference record which contains information for adding a record to one of the supported cross reference lists.

listsize — A parameter returned by the MuList...() calls which represents the size of the returned MapReference list.

DESCRIPTION

The Chart Manager categorizes certain items which need to be unique, and internally represented as integers, and provides services to convert between their internal representation and an external (string) representation. The items which fall under this method include: MapType, MapSubType, Projection-Type, FeatureType and FeatureSubType. These items normally use their internal representation for communication with the Chart Manager (for example, see MProdAtts(3Map)).

Categories can be externally represented either as a string, or as an acronym. The Mu....ToString() routines provide a string representation for the internal category id. The Mu....ToAcronym() routines provide an acronym representation for the internal category id. Both routines return pointers to allocated character strings. C programmers should free these strings using MFree. If an error occurs while retrieving the references, NULL is returned.

The MuStringTo....() routines provide a mechanism for converting an external string representation for a category to its unique internal representation. The returned value can then be used in the appropriate fields in records sent to the Chart Manager. These strings can be either the string or acronym representation for an item in a category. You need to provide only enough of the string to uniquely distinguish the item from other items in the category. The format "string (acronym)" is also accepted. If the string match fails to match any item in the category, or if the matching algorithm detects an ambiguity, then the value UnknownValue is returned. Otherwise, the internal id for that item is returned.

The MuAdd...() routines enable the calling module to add the indicated value to the value list. If the value already is present on the list, it will not be added again. Since the caller must have write access to certain files, those files may have their protection set to disable this capability.

The MuList....() routines return a list of structures for all items in a category. Each record in the MapReference list contains an internal representation, a string representation, and an

acronym representation. The list size is returned in the listsize parameter. When done with the list, C users must free it via a call to MFree.

The MuSubType...() and MuSubFeature...() routines require an additional parameter, the type. Map subtypes are

categorized by MapType, and must be referenced as such.
When type is set to AnyMap, then this subtype is valid for all map classes (in the case of MuAddSubType()), and refers to the first entry of this type for the other subtype routines.

Similarly, feature subtypes are categorized by FeatureType, and must be referenced as such. When type is set to Any- Feature, then this subtype is valid for all feature classes (in the case of MuAddSubFeature()), and refers to the first entry of this type for other subfeature routines.

RETURN

The Mu...ToString() and Mu...ToAcronym() routines return a pointer to a string which must be freed using MFree. Likewise, the MuList...() routines return a pointer to a list of MapReference structures, which must be freed using MFree.

The MuStringTo...() routines return an internally encoded integer value which uniquely represents the string. The value UnknownValue is returned in cases where the the matching algorithm fails. The MuAdd...() routines return NoError on success, and an error status indicating the cause of failure otherwise.

STRUCTURES

C Interface

```
typedef struct _MReference {
        union {
          unsigned int internal;
          char code[4];
        }     value;
        char  name[TYPE_LENGTH+1];
                char  acronym[ACRONYM_LENGTH+1];
        unsigned int    typevalue;
} MapReference;
```

FILES

${MapClassPath}/MapType.xrf
The cross reference file for the MapType category.

${MapClassPath}/MapSubType.xrf
The cross reference file for the MapSubType category.

${MapClassPath}/MapProjection.xrf
The cross reference file for the ProjectionType

category.

${MapClassPath}/MapFeatures.xrf
The cross reference file for the FeatureType category.

${MapClassPath}/MapSubFeatures.xrf
The cross reference file for the FeatureSubType category.

ENVIRONMENT

MapClassPath
This path specifies the location of Chart Manager cross reference files. It must be defined in order to use the MuReference() routines.

ERRORS

Returned by the MuAdd...() routines:

BadValueError
A supplied address pointer is invalid.

BadRecord
The cross reference file contains a bad or illegally formatted record.

NoEnvironment
The environment variable "MapClassPath" is not defined.

NoSuchFile
One of the above named files does not exist.

SEE ALSO

MError(3C), MFeatAtts(3Map), MProdAtts(3Map)

MuUnits

FUNCTION

Chart Manager units conversion macros.

SYNTAX

C Interface

```
#include<M/MuUnits.h>

FLOAT DegreesToRadians(arg)
FLOAT RadiansToDegrees(arg)
FLOAT NauticalToMeters(arg)
FLOAT NauticalToKilometers(arg)
FLOAT NauticalToMiles(arg)
FLOAT NauticalToDegrees(arg)
FLOAT MetersToNautical(arg)
FLOAT MetersToFeet(arg)
FLOAT KilometersToNautical(arg)
FLOAT KilometersToMiles(arg)
FLOAT FeetToMeters(arg)
FLOAT MilesToNautical(arg))
FLOAT MilesToKilometers(arg)
FLOAT MilesToMeters(arg)
```

ARGUMENTS

arg	The argument passed is cast to a FLOAT so that any scalar value will be accepted.

DESCRIPTION

MuUnits utilities provide a set of standard conversion routines for converting units.

RETURN STATUS

All arguments are returned as a FLOAT.

MuUtmPosn

FUNCTION

Chart Manager UTM position string utility routines.

SYNTAX

C Interface

```
        void MuUtmToString ( string, utm, format)
    char  *string;    /*Output*/

    UTM_COORD  *utm;     /*Input*/ int format;   /*Input*/

int MuStringToUtm ( string, utm) char  *string;        /*Input*/ UTM_COORD
    *utm;     /*Output*/
```

ARGUMENTS

string — The string representation of a civilian Universal Transverse Mercator (UTM) formatted position. The syntax of this string is:

<zone>[,][<spheroid>][,]<northing>[,]<easting>

where

<zone> is the Universal Transverse Mercator (UTM) zone.
<spheroid> is the UTM spheroid. This field is optional, and the resultant spheroid will be returned from the other information if it is not supplied. If no spheroid is to be supplied, do not include commas.
<easting> is the absolute easting in meters. This is a floating point value.

<northing> is the absolute northing in meters. This is also a floating point value.

utm — The UTM coordinate record.

format — The format for the printed string. Valid values are: UtmNoSpheroid and UtmPlusSpheroid.

DESCRIPTION

The MuUtmPosn routines provide conversions from UTM values to a printable string, and vice-versa. The routine MuUtmTo- String() returns a string representation of the input position. The parameter string must be large enough to accept the output (at least 25 characters). Two formats are supported: UtmNoSpheroid and UtmPlusSpheroid. If

UtmNoS- pheroid format is requested, then the UTM position is printed out without the spheroid value used in the transformation. If UtmPlusSpheroid format is requested, then the spheroid used in the transformation is printed out after the

zone.
The routine MuStringToUtm() parses the input string and returns a UTM COORD record value if the position string is valid. This routine returns the value CMNR SUCCESS if the conversion is successful, and returns one of a number of possible error codes if the conversion fails. The error codes are explained in detail in the MuConvert(3Mu) manual page. The MuStringToUtm() routine calls MuUtmGeo() to validate the parsed position. The MuStringToUtm() call is guaranteed to return an invalid UTM coordinate if the call fails for some reason. Likewise, the MuUtmToString() call will return asterisks if the input UTM coordinate is invalid.

SEE ALSO

MuConvert(3Mu)

NOTES

Both routines should be able to support user defined format strings. This would circumvent the need to define and support a new format each time one is required.